OPENMP® OFFLOAD CAPABILITIES IN ONEAPI HPC TOOLKIT

Jeongnim Kim, PhD
Principal Engineer
Intel Corporation
jeongnim.kim@intel.com
June 24, 2020

Agenda

- OpenMP® for accelators
- Managing data movement
- Expressing Parallelisms
  - Data parallelism
  - Hierarchical parallelism
  - CPU-GPU parallelism
- Coming-soon features
- Conclusions

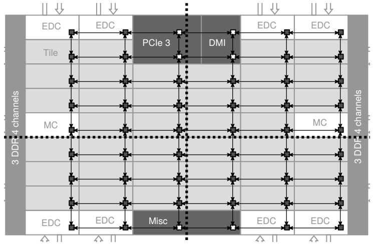
Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

OpenMP® for developing parallel applications

https://www.openmp.org/

a *portable*, *scalable* model that gives programmers a simple and *flexible* interface for developing *parallel* applications for a wide range of platforms – Wikipedia

SGI/Cray Origin 2000, NCSA, 1997



Intel KNL, Theta, ALCF

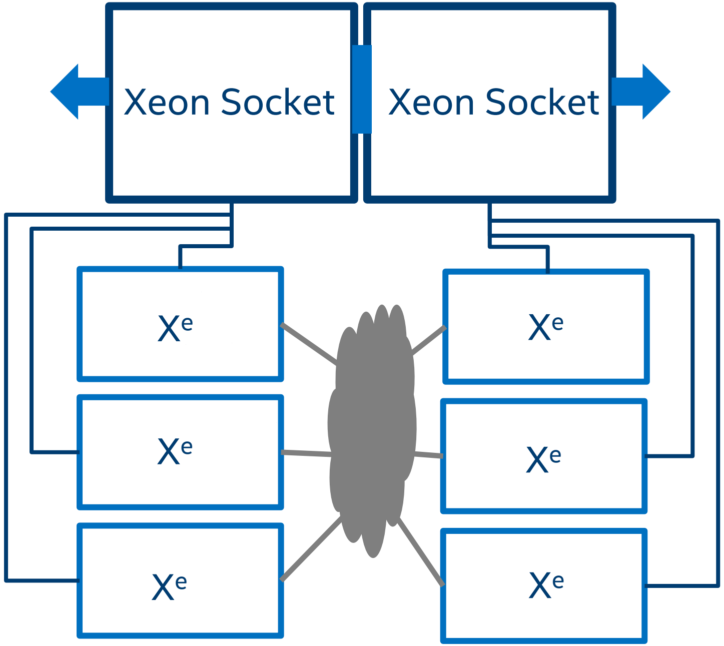
Intel® Xeon + Xe, 2021

Resources

- ALCF OpenMP training
- https://github.com/UoB-HPC/openmp-tutorial
- oneAPI webinar on OpenMP, Xinmin Tian, Intel

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

OpenMP® APIs for heterogeneous systems



Schematics of Aurora Supernode

Provide a set of directives to instruct the compiler and runtime to offload a block of code to the device.

Allow applications to exploit much increased compute density and BW of accelerators, such as X^e GPU.

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Reminders for the developers of parallel codes on heterogeneous platforms with discrete GPUs

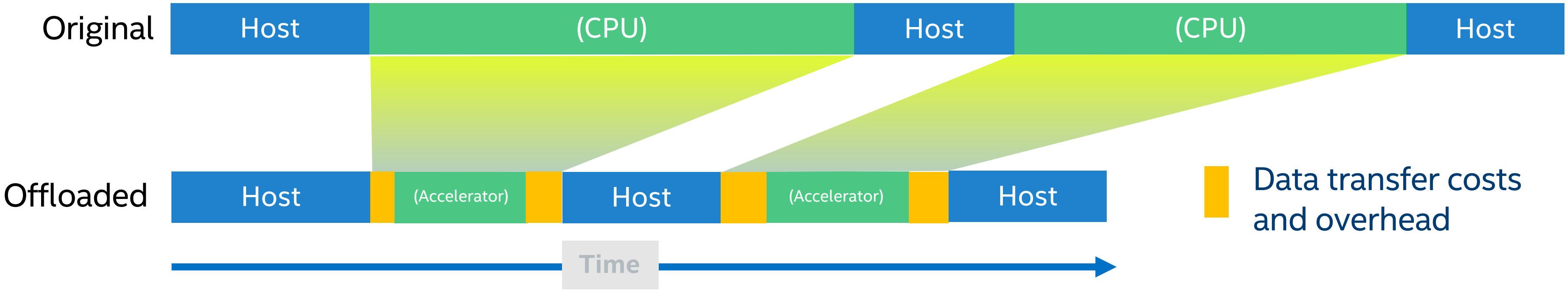
- Massively parallel but simple compute engines
  - 72-EU Gen9: 72 EU *7 threads*32 SIMD= 16128
  - Expect big increases for future X^e
- Thread blocks, block of threads and SIMD (WARP, wavefront)
  - Memory model, forward progress guarantee, synchronization
- Distinct memory spaces of host and GPUs
  - Where the data are allocated and reside and how to move are critical
  - Unified Shared/Virtual Memory removes the need for the programmers to explicitly move data but does not remove data movement
- Heterogeneous and hierarchical memory
  - Memory BW: host-host, host-GPU, HBM/DDR on GPUs, Cache

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Offload Where it Pays Off the Most

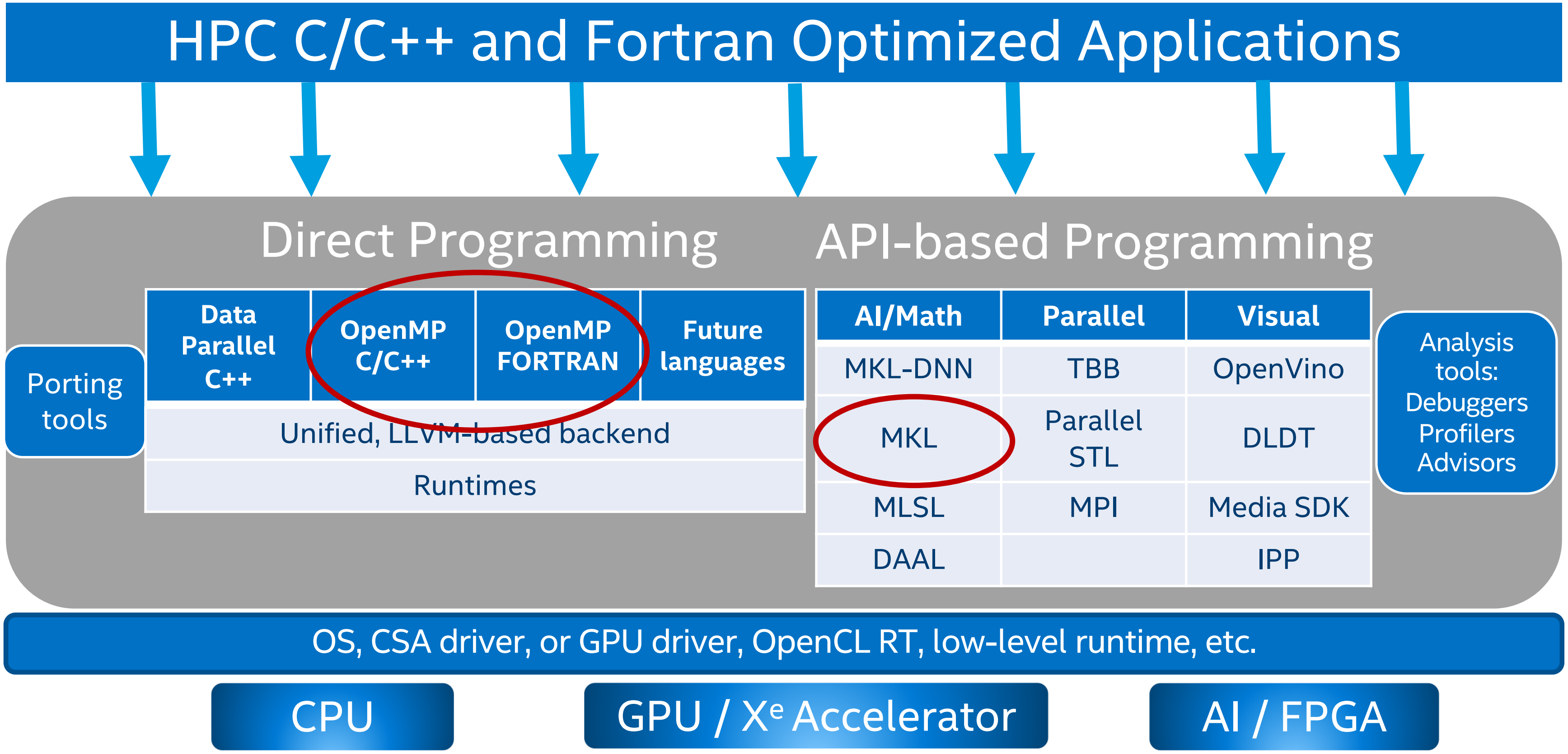
Design your code to efficiently offload to accelerators

- Determine if your code would benefit from offload to accelerator – even before you have the hardware
- Identify the opportunities to offload
- Project performance on accelerators
- Estimate overhead from data transfers and kernel launch costs
- Pinpoint accelerator performance bottlenecks (memory, cache, compute and data transfer)
- Follow good SIMD guidelines (e.g. avoid branch divergence and gathers/scatters)



Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Intel® oneAPI HPC Toolkit (beta)



Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

OpenMP® using oneAPI® compilers

Based on beta07 release http://www.oneapi.com

- Download and install oneAPI HPC Toolkit
- Setup oneAPI environment

```
$source /opt/intel/inteloneapi/setvars.sh
```

- Compile a C++ application OpenMP target (offload)

```
$icpx –fiopenmp –fopenmp-targets=spir64 test.cpp
```

```
$icpc –qnextgen –fiopenmp –fopenmp-targets=spir64 test.cpp
```

- Compile an application using oneMKL

```
$icx –I${MKLROOT}/include –DMKL_ILP64 –m64 –fiopenmp
–fopenmp-targets=spir64 –c <file>.c{pp} –o <file>.o
$icx <file>.o –fiopenmp –fopenmp-targets=spir64 –lOpenCL
-L${MKLROOT}/lib/intel64 -lmkl_intel_ilp64 -lmkl_intel_thread \
-lmkl_core -lpthread -ldl -lm -o <file>
```

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

OpenMP® using oneAPI® compilers

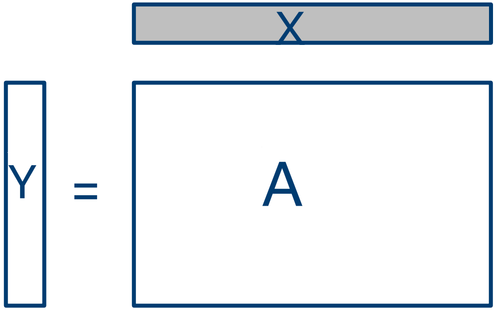
- Useful environments for a run

```
LIBOMPTARGET_DEBUG=<int>
LIBOMPTARGET_PROFILE=T
OMP_TARGET_OFFLOAD=MANDATORY|DISABLED|DEFAULT
```

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Matrix-vector multiplication (GEMV)

```
size_t N=1024;
size_t M=1048576;
Matrix<float> A(N,M);
Vector<float> X(M), Y(N);

// initialization

for(int i=0; i<N; ++i) {
  float sum{};
  for(int j=0; j<M; ++j) {
    sum += A[i][j]*X[j];
  }
  Y[i]=sum;
}
```

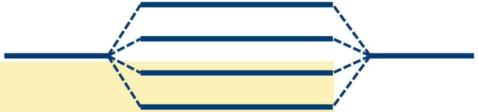


Using pseduo codes inspired and based on miniapps, Ye Luo (ANL), QMPCACK ECP
https://github.com/QMCPACK/miniqmc/

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Parallel Matrix-vector multiplication

```
size_t N=1024;
size_t M=1048576;
Matrix<float> A(N,M);
Vector<float> X(M), Y(N);

// initialization

for(int i=0; i<N; ++i) {
  float sum{};
  for(int j=0; j<M; ++j) {
    sum += A[i][j]*X[j];
  }
  Y[i]=sum;
}
```



```
#pragma omp parallel for
  for(int i=0; i<N; ++i) {
    float sum{};

    for(int j=0; j<M; ++j) {
      sum += A[i][j]*X[j];
    }
    Y[i]=sum;
  }
```

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Parallel-SIMD Matrix-vector multiplication

```
size_t N=1024;
size_t M=1048576;
Matrix<float> A(N,M);
Vector<float> X(M), Y(N);

// initialization

for(int i=0; i<N; ++i) {
  float sum{};
  for(int j=0; j<M; ++j) {
    sum += A[i][j]*X[j];
  }
  Y[i]=sum;
}
```

```
#pragma omp parallel for
  for(int i=0; i<N; ++i) {
    float sum{};
#pragma omp simd reduction(+:sum)
    for(int j=0; j<M; ++j) {
      sum += A[i][j]*X[j];
    }
    Y[i]=sum;
  }
```

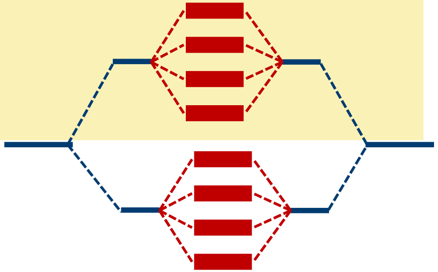
Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Compose your parallel problem

OMP_NESTED=TRUE

```
#pragma omp parallel
{
#pragma omp for nowait
  for(int i=0; i<N; ++i) {
    float sum{};
#pragma omp simd reduction(+:sum)
    for(int j=0; j<M; ++j) {
      sum += A[i][j]*X[j];
    }
    Y[i]=sum;
  }

  // do many more

}
```

```
#pragma omp parallel
{
#pragma omp for nowait
  for(int i=0; i<N; ++i) {
    float sum{};
#pragma omp parallel for simd reduction(+:sum)
    for(int j=0; j<M; ++j) {
      sum += A[i][j]*X[j];
    }
    Y[i]=sum;
  }

  // do many more

}
```



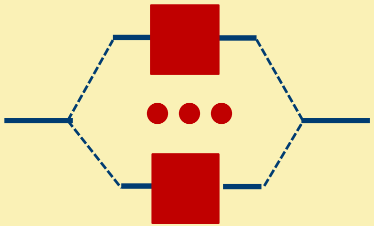
Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

GEMV with OpenMP® 4.5

1. Transfer control of execution to a device
2. Map A and X **to** a device
3. Map Y **from** a device to host
4. Create teams of threads
5. Distribute the loop
6. Execution the loop in parallel
7. Reduce sum within a team
8. Assign the sum to Y

```
size_t N=1024;
size_t M=1048576;
Matrix<float> A(N,M);
Vector<float> X(M), Y(N);

// initialization

for(int i=0; i<N; ++i) {
  float sum{};
  for(int j=0; j<M; ++j) {
    sum += A[i][j]*X[j];
  }
  Y[i]=sum;
}
```

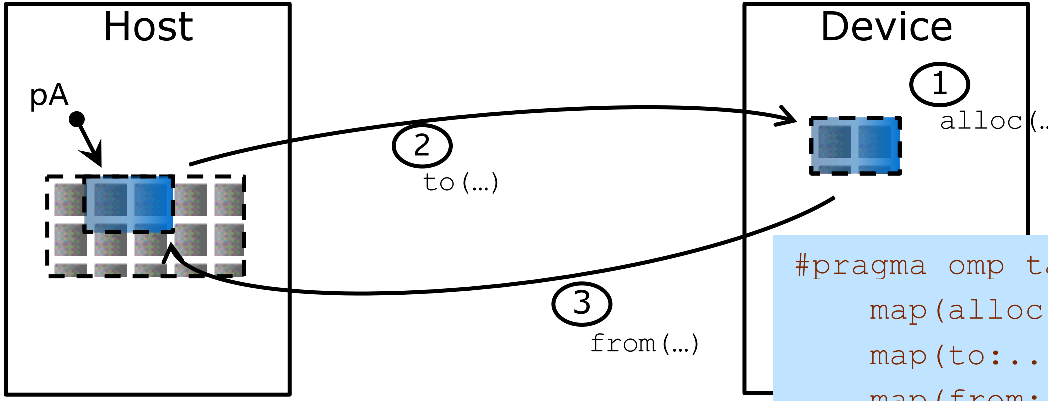
```
Matrix<float> A(N,M);
Vector<float> X(M), Y(N);


float *pA=A.data(), *pX=X.data(), *pY=Y.data();
#pragma omp target map(to:pA[0:N*M],pX[0:M]) map(from:pY[0:N])
{
#pragma omp teams distribute
  for(int i=0; i<N; ++i) {
    float sum{};
#pragma omp parallel for simd reduction(+:sum)
    for(int j=0; j<M; ++j) {
      sum += pA[i*M+j]*pX[j];
    }
    pY[i]=sum;
  }
}
```



Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Offloading and Device Data Mapping

- Use *target* construct to
  - Transfer control from the host to the target device
  - Map variables between the host and target device data environments




```
#pragma omp target \
    map(alloc:...) \
    map(to:...) \
    map(from:...)
{ ... }
```

- Host thread waits until offloaded region is completed
  - Use other OpenMP tasks for asynchronous execution
- The **`map`** clauses determine how an *original variable* in a data environment is mapped to a *corresponding variable* in a device data environment

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Data management

- Device allocator for the data exclusive accessed by a device

```
int deviceId= … ; // query device id
int *a = (int *)omp_target_alloc(1024, deviceId);
<use a>
omp_target_free(a, deviceId);
```

- Target data enter/exit and update

```
int A[N], B[N];
#pragma omp target enter data map(alloc:B)  map(to:A)
// do a lot of work with A & B
#pragma omp target update(A)
// do more on a device and host with new A
#pragma omp exit data map(from:A)
```

- Allocator specializations to reduce clutter and optimize data transfers

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Maximizing data parallelism

- Same tasks/computations performed on subsets of the same data
- Synchronous computations with no or minimal branches
- Increasing gain with larger data sets

```
#pragma omp teams distribute
  for(int i=0; i<N; ++i) {
    float sum{};
#pragma omp parallel for simd reduction(+:sum)
    for(int j=0; j<M; ++j) {
      sum += pA[i*M+j]*pX[j];
    }
    pY[i]=sum;
  }
```

```
#pragma omp teams distribute parallel for simd collapse(2)
for(int i=0; i<N; ++i)
  for(int j=0; j<N; ++j)
    for(int k=0; k<N; ++k) {

       Body(i,j,k);

    }
```

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Hierarchical parallelism on a GPU

```
#pragma omp target is_device_ptr(pA,pX,pZ) map(from:pY)
{
#pragma omp teams distribute
  for(int i=0; i<N; ++i) {
    float sum{};
#pragma omp parallel for simd reduction(+:sum)
    for(int j=0; j<M; ++j) {
      sum += pA[i*M+j]*pX[j];
    }
    pY[i]=sum;
#pragma omp parallel for simd
    for(int j=0; j<M; ++j) {
      pZ[j]+=sum*pX[j];
    }
  }
}
```

- Nested loops with shared variables
- Limited parallelism
- Data dependencies within a team
- Potential data reuse
- But, use with care!

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Mixing host and GPU parallelism

```
#pragma omp parallel
{
  //per thread allocations

  #pragma omp target is_device_ptr(pA,pX,pZ) map(from:pY)
  {
  #pragma omp teams distribute
    for(int i=0; i<N; ++i) {
      float sum{};
  #pragma omp parallel for simd reduction(+:sum)
      for(int j=0; j<M; ++j) {
        sum += pA[i*M+j]*pX[j];
      }
      pY[i]=sum;
  #pragma omp parallel for simd
      for(int j=0; j<M; ++j) {
        pZ[j]+=sum*pX[j];
      }
    }
  }

}
```

```
#pragma omp target nowait
{


}

do_other_things();

#pragma omp taskwait
```

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Unified Shared Memory Support

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define SIZE 1024
#pragma omp requires unified_shared_memory
int main() {
  int deviceId = (omp_get_num_devices() > 0) ? omp_get_default_device() : omp_get_initial_device();
  int *a = (int *)omp_target_alloc(SIZE, deviceId);
  int *b = (int *)omp_target_alloc(SIZE, deviceId);
  for (int i = 0; i < SIZE; i++) {
    a[i] = i;     b[i] = SIZE - i;
  }
#pragma omp target parallel for
  for (int i = 0; i < SIZE; i++) {
    a[i] += b[i];
  }

  for (int i = 0; i < SIZE; i++) {
    if (a[i] != SIZE) {
      printf("%s failed\n", __func__);  return EXIT_FAILURE;
    }
  }
  omp_target_free(a, deviceId);
  omp_target_free(b, deviceId);
  printf("%s passed\n", __func__);
  return EXIT_SUCCESS;
}
```

Adding USM support via managed memory allocator

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

OpenMP* and DPC++ Composability

```
#include <CL/sycl.hpp>
#include <array>
#include <iostream>
float computePi(unsigned N) {
float Pi;
#pragma omp target map(from : Pi)
#pragma omp parallel for reduction(+ : Pi)
  for (unsigned I = 0; I < N; ++I) {
    float T = (I + 0.5f) / N;
    Pi += 4.0f / (1.0 + T * T);
  }
  return Pi / N;
}
// DPC++ Code
void iota(float *A, unsigned N) {
  cl::sycl::range<1> R(N);
  cl::sycl::buffer<int,1> X(A, R);
  cl::sycl::queue().submit([&](cl::sycl::handler &cgh) {
    auto Y = X.template get_access<cl::sycl::access::mode::write>(cgh);
    cgh.parallel_for<class Iota>(R, [=](cl::sycl::id<1> idx) {
       Y[idx] = idx;
    });
  });
}
```

OpenMP offloading code

DPC++ code

```
int main() {
  std::array<int, 1024u> V;
  float Pi;
#pragma omp parallel sections
  {
#pragma omp section
    iota(V.data(), V.size());
#pragma omp section
    Pi = computePi(8192u);
  }

  std::cout << "V[512] = " << V[512] << std::endl;
  std::cout << "Pi = " << Pi << std::endl;
  return 0;
}
```

```
xtian@scsel-cfl-02:~/temp$ icpx -fiopenmp -fopenmp-targets=spir64 -fsycl  compos.cpp -o run.y
xtian@scsel-cfl-02:~/temp$ OMP_TARGET_OFFLOAD=mandatory ./run.y
V[512] = 512
Pi = 3.14159
```

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

oneMKL C OpenMP offload Example (GEMM)

```
#include "mkl.h"
#include "mkl_omp_offload.h"
int main() {
    MKL_INT m = 10, n = 6, k = 8, lda = 12, ldb = 8, ldc = 10;
    MKL_INT sizea = lda * k, sizeb = ldb * n, sizec = ldc * n;
    double alpha = 1.0, beta = 0.0;
    // Allocate matrices
    double *A = (double *)mkl_malloc(sizeof(double) * sizea, 64);
    double *B = (double *)mkl_malloc(sizeof(double) * sizeb, 64);
    double *C = (double *)mkl_malloc(sizeof(double) * sizec, 64);
    // initialize matrices
    …
#pragma omp target data map(to:A[0:sizea],B[0:sizeb]) map(tofrom:C[0:siz
    {
#pragma omp target variant dispatch use_device_ptr(A, B, C) [nowait]
        {
            // Compute C = A * B on GPU
            cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, m, n
                        alpha, A, lda, B, ldb, beta, C, ldc);
        }
    }
```

Specific header file for oneMKL OpenMP offload

Use target variant dispatch to notify GPU computation is requested

List all device memory pointer in the use_device_ptr clause

Optional nowait clause for asynchronous execution, use omp taskwait for synchronization

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

Get Started with oneAPI Today!

Resources

Intel® DevCloud	**Start in the Cloud** - No Download, No Installation, No Setup – Sign up here - software.intel.com/devcloud/oneAPI
oneAPI Toolkits	**Develop On-Prem –** Download & Develop - Get them here - software.intel.com/oneAPI
Industry Support	**Break Free Now –** CodePlay* Contributes Data Parallel C++ Support for NVIDIA* GPU github.com/intel/llvm
oneAPI Specification	**Join the Initiative** - Cross-industry, open, standards-based unified programming model across architectures – Learn more here - oneapi.com

Learn Data Parallel C++

Build XPU Applications
Intel® Xeon, FPGA, Integrated Graphics GPUs & X^e GPU, DG1**

Prototype Your Project
Evaluate Workloads

Learn about Intel®
oneAPI Toolkits

Interoperable with MPI, OpenMP, Fortran, C/++

oneAPI™

Gaining Momentum!
oneAPI 0.8 Now Available!

** Available under NDA on Intel® DevCloud

Copyright © Intel Corporation 2020
*Other names and brands may be claimed as the property of others.

intel Software

Notices & Disclaimers

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

The benchmark results reported herein may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Copyright © 2020, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and OpenVINO are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries. Khronos® is a registered trademark and SYCL is a trademark of the Khronos Group, Inc.

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