Intel® OpenMP* Compilers for CPUs and GPUs

Xinmin Tian, Intel Corporation

OpenMPCon 2021 | Bristol, UK | 13-15 September 2021 | 6th OpenMP Developers Conference

intel®

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

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. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

AGENDA

- Multi-tile X^e GPU Hierarchy: Recap
- oneAPI Software Stack
- Just-In-Time (JIT) Compilation Flow
- OpenMP SIMD for GPUs
- Unified Shared Memory (USM)
- OpenMP and DPC++ Composability
- New Features (declare mapper, class member functor, function pointer in offload region)
- Multi-GPU and Multi-Tile Support
- Asynchronous Offloading
- Optimization Report
- Summary

Xe HP Multi-tile GPU

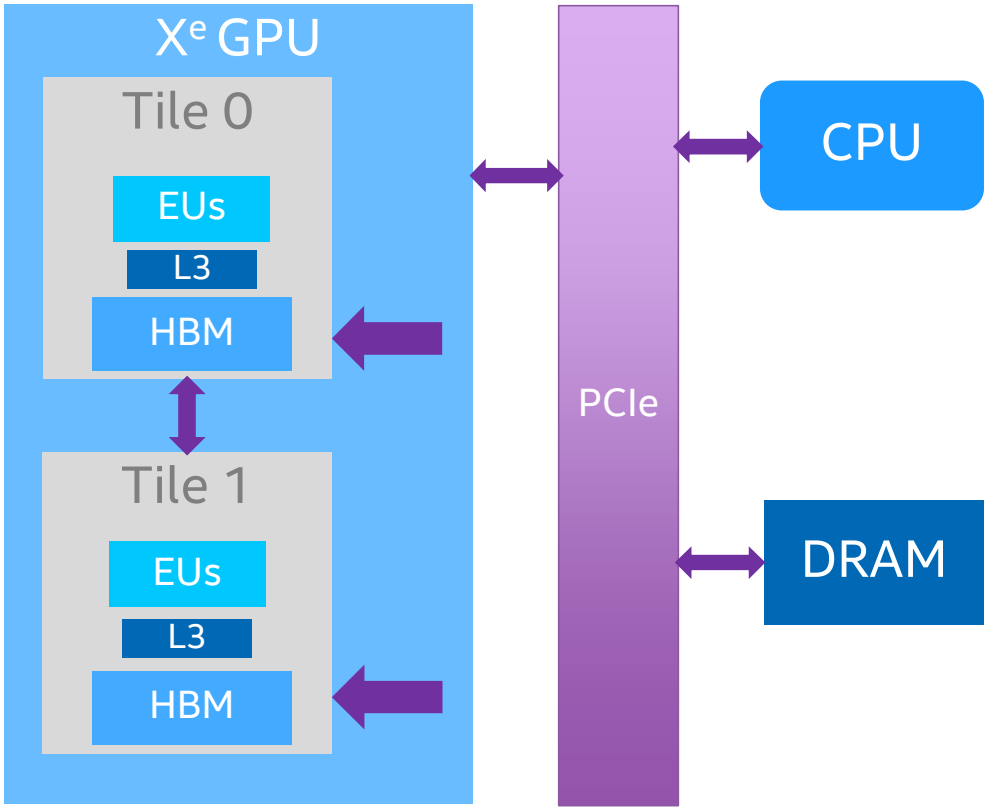
David Blythe at HotChips'2020
Raja Koduri at Intel Architecture Day'2021

1-Tile
>10 FP32 TFLOPS

2-Tile
>20 FP32 TFLOPS

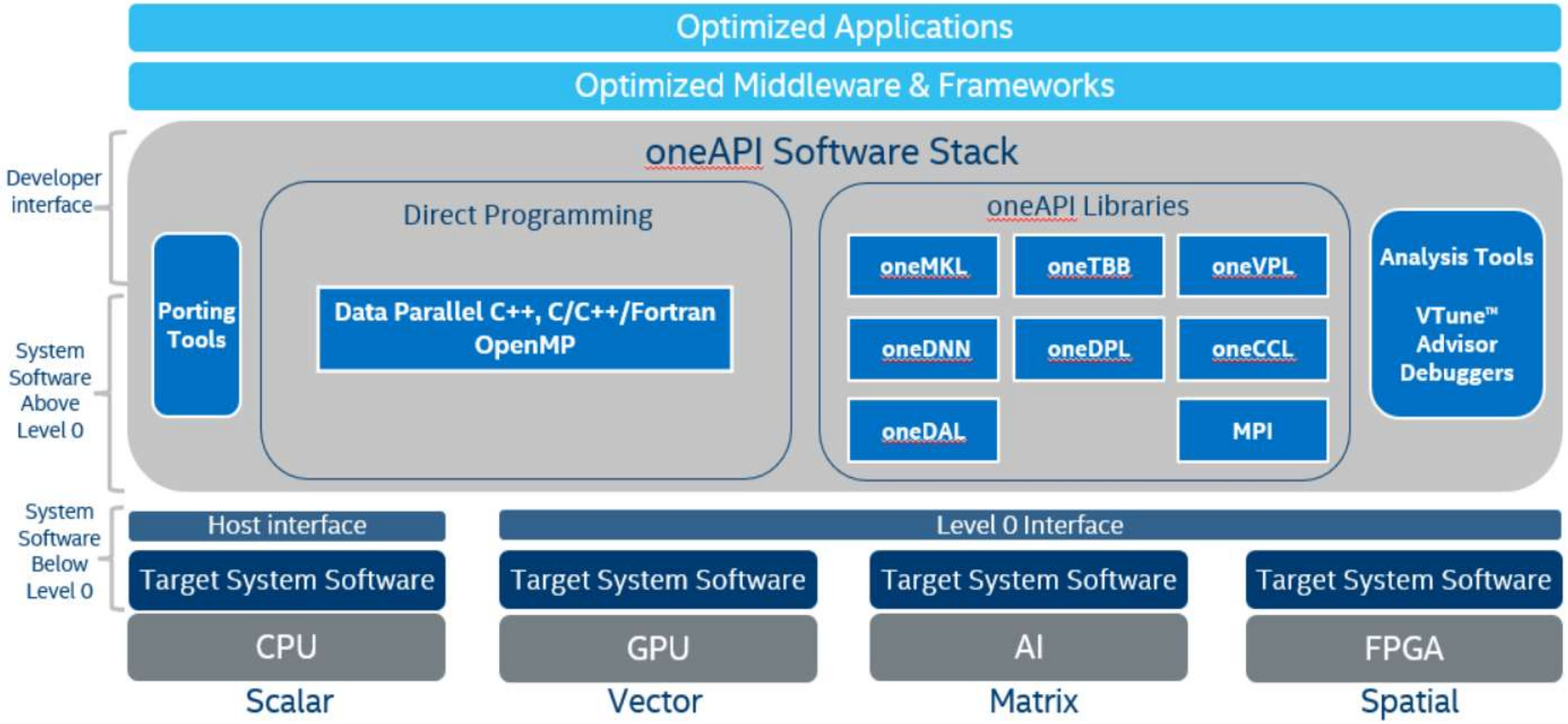
4-Tile
>40 FP32 TFLOPS

Xe GPU Architecture Hierarchy

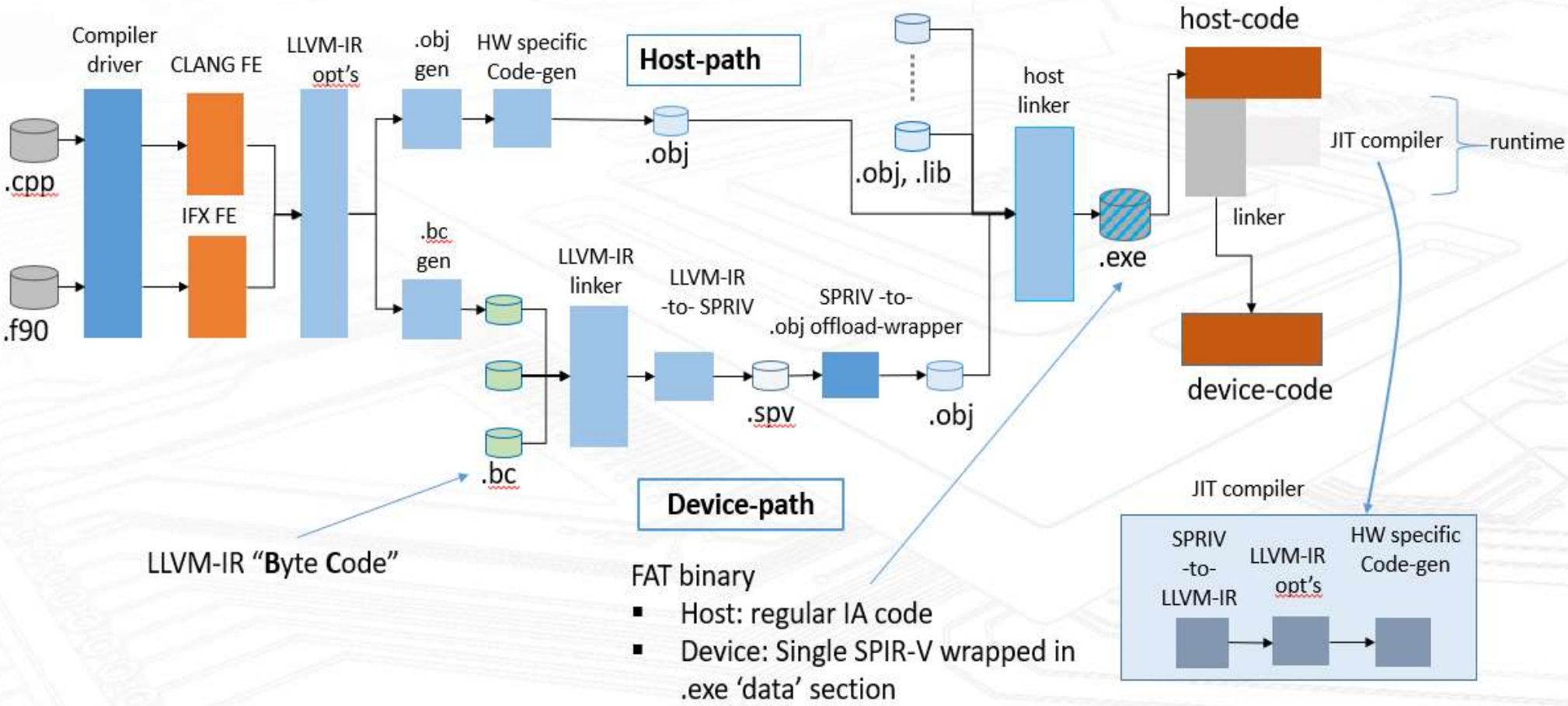


- Tiles are independent
  - No global scheduler (each tile has its own scheduler)
  - Tiles can work concurrently
  - No Global Commands affecting all tiles
  - No global state
- Tiles can communicate over memory
  - Synchronization between tiles uses memory and GPU semaphores

oneAPI Software Stack for HPC and AI Applications



Just-In-Time (JIT) Compilation Flow



LLVM-IR "Byte Code"

FAT binary

- Host: regular IA code
- Device: Single SPIR-V wrapped in .exe 'data' section

OpenMP SIMD for GPUs

```
#pragma omp target enter data map( alloc:a[0:TOTAL_SIZE] )
#pragma omp target enter data map( alloc:b[0:TOTAL_SIZE] )
#pragma omp target enter data map( alloc:c[0:TOTAL_SIZE] )
#pragma omp target update to(a[0:TOTAL_SIZE])
#pragma omp target update to(b[0:TOTAL_SIZE])

const int no_max_rep = 400;
double time = omp_get_wtime();
for ( int irep = 0; irep < no_max_rep; ++irep ) {
  #pragma omp target teams distribute parallel for
  for ( int isimd = 0; isimd < TOTAL_SIZE; isimd += SIMD_SIZE<<2) {
    #pragma omp simd simdlen(32)
    for (int ilane = 0; ilane < SIMD_SIZE<<2; ++ilane) {
      const int index = isimd + ilane;
      c[index] = a[index] + b[index];
    }
  }
}
time = omp_get_wtime() - time;
time = time/no_max_rep;
 … … … …
#pragma omp target exit data map( release:a[0:TOTAL_SIZE] )
#pragma omp target exit data map( release:b[0:TOTAL_SIZE] )
#pragma omp target exit data map( release:c[0:TOTAL_SIZE] )
```

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

- omp_target_alloc_host(…)
- omp_target_alloc_shared(…)
- omp_target_alloc_device(…)

DPC++ and OpenMP Composability

OpenMP offloading code

```
#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

```
// 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;
    });
  });
}
```

```
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
```

OpenMP and DPC++/SYCL USM Composability

```
#include <CL/sycl.hpp>
#include <omp.h>
#include <algorithm>
#include <iostream>

extern "C" void *omp_target_get_context(int);
#ifndef LEVEL0
#pragma omp requires unified_shared_memory
#endif

int main() {
  const unsigned Size = 200;
  int D = omp_get_default_device();
#ifndef LEVEL0
  cl::sycl::queue Q(
      cl::sycl::context(static_cast<cl_context>(omp_target_get_context(D))),
      cl::sycl::gpu_selector());
#else
  cl::sycl::queue Q;
#endif
  std::cout << "SYCL: Running on "
            << Q.get_device().
               get_info<cl::sycl::info::device::name>() << "\n";
  if (!Q.get_device().
      get_info<cl::sycl::info::device::usm_shared_allocations>()) {
    std::cout << "SYCL: USM is not available\n";
    return 0;
  }
  auto validate = [](int *Data) {
    for (unsigned I = 0; I < Size; ++I)
      if (Data[I] != 100 + I) return "failed";
    return "passed";
  };
```

```
  auto testOmp = [&](int *Data) {
    std::fill_n(Data, Size, -1);
    #pragma omp target parallel for device(D)
    for (unsigned I = 0; I < Size; ++I) {
      Data[I] = 100 + I;
    }
    return validate(Data);
  };
  auto testDpc = [&](int *Data) {
    std::fill_n(Data, Size, -1);
    Q.submit([&](cl::sycl::handler &cgh) {
      cgh.parallel_for<class K>
          (cl::sycl::range<1>(Size), [=](cl::sycl::id<1> I) {
        Data[I] = 100 + I;
      });
    }); Q.wait();
    return validate(Data);
  };

  int *ompMem = (int*)omp_target_alloc_shared(Size*sizeof(int), D);
  int *dpcMem = cl::sycl::malloc_shared<int>(Size, Q);
  std::cout<<"SYCL and OMP memory: "<<testDpc(ompMem) << "\n";
  std::cout<<"OMP and OMP memory:  "<<testOmp(ompMem) << "\n";
  std::cout<<"OMP and SYCL memory: "<<testOmp(dpcMem) << "\n";
  std::cout<<"SYCL and SYCL memory: "<<testDpc(dpcMem) << "\n";
  omp_target_free(ompMem, D);
  cl::sycl::free(dpcMem, Q);
  return 0;
}
```

Class Member Functor in Offload Region

```
#include <bits/stdc++.h>
using namespace std;

// A Functor
class inc
{
private:
    int num;
public:
    inc(int n) : num(n) {  }
    int operator () (int arr_num) const {
        return num + arr_num;
    }
};
```

```
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    int add5 = 5;
    inc a_inc(add5);

    #pragma omp target teams distribute \
           parallel for map(arr[0:n]) map(to: a_inc)
    for (int k = 0; k < n; k++) {
       arr[k] = arr[k] + a_inc(k);
    }

    for (int i=0; i<n; i++) cout << arr[i] << " ";
    cout << "\n" << "Done ......\n";
}
```

Function Pointer Support in Offload Region

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

#pragma omp begin declare target
int foo(int y)
{
   printf("called from device, y = %d\n", y);
   return y;
}
#pragma omp end declare target
```

```c
int main()
{
  int x = 0;
  int y = 100;
  int (*fptr)(int) = foo;
  #pragma omp target teams \
        distribute parallel for \
        firstprivate(y) reduction(+: x)
  for (int k = 0; k < 16; k++) {
      x = x + fptr(y + k);
  }
  printf("Output x = %d\n", x);
}
```

User Defined Mapper (declare mapper)

```
extern "C" int printf(const char *, ...);
struct C {   int num;   int *arr;  };
#pragma omp declare mapper(id: C c) map(c.num, c.arr[0:c.num])

void foo(int num, int *arr, int *arr_one) {
    int i;  C c;  c.num = num;  c.arr = arr;
    for (i=0; i<num; ++i) printf("%s%3d %s", (i==0?"In : ":""), c.arr[i], (i==num-1?"\n":""));

    #pragma omp target map(mapper(id),tofrom: c)
    { int j;   for(j=0; j < c.num; ++j)      c.arr[j] *= 2;   }
    for (i=0; i<num; ++i) printf("%s%3d %s", (i==0?"Out: ":""), c.arr[i], (i==num-1?"\n":""));

    C c_one;  c_one.num = num;  c_one.arr = arr_one;
    for (i=0; i<num; ++i) printf("%s%3d %s", (i==0?"In : ":""), c_one.arr[i], (i==num-1?"\n":""));

    #pragma omp target map(mapper(id),tofrom: c_one)
    { int j;    for(j=0; j < c_one.num; ++j) c_one.arr[j] *= 2; }
    for (i=0; i<num; ++i) printf("%s%3d %s", (i==0?"Out: ":""), c_one.arr[i], (i==num-1?"\n":""));
}
int main() {
    int arr4[] = { 1,2,4,8 };      int arr8[] = { 1,2,4,8,16,32,64,128 };
    int arr4one[] = { 1,2,4,8 };   int arr8one[] = { 1,2,4,8,16,32,64,128 };
    foo(sizeof(arr4)/sizeof(arr4[0]), arr4, arr4one);  foo(sizeof(arr8)/sizeof(arr8[0]), arr8, arr8one);
    return 0;
}
```

Multi-GPU and Multi-Tile support

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 16384
int main() {
  int numDevices = omp_get_num_devices();
  int numThreads = 4 * numDevices;
  int *a = new int[N]; int *b = new int[N]; int *c = new int[N];
  for (int i = 0; i < N; i++) {   a[i] = i; b[i] = N - i;  }
  printf("Testing offload with %d devices, %d threads\n", numDevices, numThreads);
#pragma omp parallel num_threads(numThreads)
  {
    int threadNum = omp_get_thread_num();
    int deviceNum = threadNum % numDevices;
    int begin = threadNum * N / numThreads;
    int count = N / numThreads;
#pragma omp target teams distribute parallel for device(deviceNum) \
                        map(a[begin:count]) map(b[begin:count]) map(c[begin:count])
    for (int i = 0; i < count; i++) {
      c[begin + i] = a[begin + i] + b[begin + i];
    }
  }
  for (int i = 0; i < N; i++) {
    if (N != c[i]) { printf("FAIL\n"); return EXIT_FAILURE; }
  }
  printf("PASS\n"); delete[] a; delete[] b; delete[] c;
  return EXIT_SUCCESS;
}
```

SpecACCEL/514.pomriq: Subdevice Usage

```
//Declaration
#define NT 2
struct mptr {
    float * p[NT];
};
struct sptr {
    kValues * p[NT];
};
  mptr device_ptr_x;
  mptr device_ptr_y;
  mptr device_ptr_z;
  mptr device_ptr_Qr;
  mptr device_ptr_Qi;
  sptr device_ptr_kVals;

  const int device_id = omp_get_default_device();
//Initialization
  int TILE_SIZE = numX/NT;
  for ( int igpu = 0; igpu < NT; ++igpu ) {

        device_ptr_x.p[igpu] = (float*)malloc( TILE_SIZE * sizeof(float) );
        device_ptr_y.p[igpu] = (float*)malloc( TILE_SIZE * sizeof(float) );
        device_ptr_z.p[igpu] = (float*)malloc( TILE_SIZE * sizeof(float) );
        device_ptr_Qr.p[igpu] = (float*)malloc( TILE_SIZE * sizeof(float) );
        device_ptr_Qi.p[igpu] = (float*)malloc( TILE_SIZE * sizeof(float) );

        for ( int k = 0; k < TILE_SIZE; ++k) {
                device_ptr_x.p[igpu][k] = x[igpu*TILE_SIZE+k];
                device_ptr_y.p[igpu][k] = y[igpu*TILE_SIZE+k];
                device_ptr_z.p[igpu][k] = z[igpu*TILE_SIZE+k];
                device_ptr_Qr.p[igpu][k] = Qr[igpu*TILE_SIZE+k];
                device_ptr_Qi.p[igpu][k] = Qi[igpu*TILE_SIZE+k];
        }
//"Transfer"
        #pragma omp target enter data map( to:device_ptr_x.p[igpu][ 0:TILE_SIZE ] )
device(device_id) subdevice(igpu)
        #pragma omp target enter data map( to:device_ptr_y.p[igpu][ 0:TILE_SIZE ] )
device(device_id) subdevice(igpu)
        #pragma omp target enter data map( to:device_ptr_z.p[igpu][ 0:TILE_SIZE ] )
device(device_id) subdevice(igpu)
        #pragma omp target enter data map( to:device_ptr_Qr.p[igpu][ 0:TILE_SIZE ] )
device(device_id) subdevice(igpu)
        #pragma omp target enter data map( to:device_ptr_Qi.p[igpu][ 0:TILE_SIZE ] )
device(device_id) subdevice(igpu)
  }
```

```
for (int k = 0; k < numK; k++) {
    kVals[k].Kx = kx[k];
    kVals[k].Ky = ky[k];
    kVals[k].Kz = kz[k];
    kVals[k].PhiMag = phiMag[k];
  }
//Same data for both Tiles
  for ( int igpu = 0; igpu < NT; ++igpu ) {
    device_ptr_kVals.p[igpu] = (struct kValues*)malloc( numK * sizeof(struct
kValues) );
    memcpy(device_ptr_kVals.p[igpu], kVals, numK * sizeof(struct kValues));

    #pragma omp target enter data map( to:device_ptr_kVals.p[igpu][ 0:numK ] )
device(device_id) subdevice(igpu)
  }
//Received result on host
  for ( int igpu = 0; igpu < NT; ++igpu ) {
    #pragma omp target update from( device_ptr_Qr.p[igpu][ 0:TILE_SIZE ] )
device(device_id) subdevice(igpu)
    #pragma omp target update from( device_ptr_Qi.p[igpu][ 0:TILE_SIZE ] )
device(device_id) subdevice(igpu)

    for ( int k = 0; k < TILE_SIZE; ++k) {
      Qr[igpu*TILE_SIZE+k] = device_ptr_Qr.p[igpu][k];
      Qi[igpu*TILE_SIZE+k] = device_ptr_Qi.p[igpu][k];
    }
  }
 //Free
    for ( int igpu = 0; igpu < NT; ++igpu ) {
    #pragma omp target exit data map( release:device_ptr_x.p[igpu][ 0:TILE_SIZE ] )
    #pragma omp target exit data map( release:device_ptr_y.p[igpu][ 0:TILE_SIZE ] )
    #pragma omp target exit data map( release:device_ptr_z.p[igpu][ 0:TILE_SIZE ] )
    #pragma omp target exit data map( release:device_ptr_Qr.p[igpu][ 0:TILE_SIZE ]
)
    #pragma omp target exit data map( release:device_ptr_Qi.p[igpu][ 0:TILE_SIZE ]
)
    #pragma omp target exit data map( release:device_ptr_kVals.p[igpu][ 0:numK ] )
  }
```

SpecACCEL/514.pomriq: Subdevice Usage

```
#pragma omp target map(to:kVals[0:numK], x[0:numX], y[0:numX], z[0:numX]), \
                    map(tofrom:Qr[0:numX], Qi[0:numX])
  {
#pragma omp teams distribute parallel for private(expArg, cosArg, sinArg)

    for (indexX = 0; indexX < numX; indexX++) {

      float QrSum = 0.0;
      float QiSum = 0.0;

#pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
      for (indexK = 0; indexK < numK; indexK++) {
        expArg = PIx2 * (kVals[indexK].Kx * x[indexX] +
        kVals[indexK].Ky * y[indexX] +
        kVals[indexK].Kz * z[indexX]);

        cosArg = cosf(expArg);
        sinArg = sinf(expArg);

        float phi = kVals[indexK].PhiMag;
        QrSum += phi * cosArg;
        QiSum += phi * sinArg;
      }

      Qr[indexX] += QrSum;
      Qi[indexX] += QiSum;
    }
  }
```

```
#pragma omp parallel for num_threads(NT)
  for ( int igpu = 0; igpu < NT; ++igpu ) {

    float * x = device_ptr_x.p[igpu];
    float * y = device_ptr_y.p[igpu];
    float * z = device_ptr_z.p[igpu];

    float * Qr = device_ptr_Qr.p[igpu];
    float * Qi = device_ptr_Qi.p[igpu];

    struct kValues * kVals = device_ptr_kVals.p[igpu];

#pragma omp target teams distribute parallel for device(device_id) \
                                subdevice(igpu) private(expArg, cosArg, sinArg)
    for (indexX = 0; indexX < TILE_SIZE; indexX++) {

      float QrSum = 0.0;
      float QiSum = 0.0;

#pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
      for (indexK = 0; indexK < numK; indexK++) {
        expArg = PIx2 * (kVals[indexK].Kx * x[indexX] +
        kVals[indexK].Ky * y[indexX] +
        kVals[indexK].Kz * z[indexX]);

        cosArg = cosf(expArg);
        sinArg = sinf(expArg);

        float phi = kVals[indexK].PhiMag;
        QrSum += phi * cosArg;
        QiSum += phi * sinArg;
      }

      Qr[indexX] += QrSum;
      Qi[indexX] += QiSum;
    }
  }
```

Asynchronous Offloading Example

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

int main() {
  int ret = 0;
#pragma omp target map(ret) nowait
  {
    for (int i = 0; i < 1000; i++)
      for (int j = 0; j < 1000; j++)
        ret--;
    if (ret <= 0)
      ret = 1;
    printf("Device ret = %d\n", ret);
  }
  printf("Before explicit offload sync: ret = %d\n", ret);

#pragma omp taskwait
  printf("After  explicit offload sync: ret = %d\n", ret);

  return 0;
}
```

```
xtian@scsel-cfl-12:$ icpx -fiopenmp -fopenmp-targets=spir64 -mllvm -vpo-paropt-enable-async-helper-thread target_nowait.cpp -o run.x
xtian@scsel-cfl-12:$ ./run.x

Before explicit offload sync: ret = 0
Device ret = 1
After  explicit offload sync: ret = 1
```

Free agent helper thread running concurrently with the initial thread

working with community to enable free agent helper thread integration

A Simple Fortran Offloading Example

```
INTEGER FUNCTION target_teams_distribute_parallel_for()
  INTEGER :: errors_bf, errors_af
  INTEGER :: i
  INTEGER, DIMENSION(ARRAY_SIZE) :: a = (/(1, i=0,ARRAY_SIZE - 1)/)
  INTEGER, DIMENSION(ARRAY_SIZE) :: b = (/(i, i=0,ARRAY_SIZE - 1)/)
  INTEGER, DIMENSION(ARRAY_SIZE) :: c = (/(2 * i, i=0,ARRAY_SIZE - 1)/)
  INTEGER :: num_teams = 0
  INTEGER, DIMENSION(ARRAY_SIZE) :: num_threads = (/(0, i=0,ARRAY_SIZE - 1)/)
  INTEGER :: alert_num_threads = 0
  CHARACTER(len=300) :: msgHelper
  OMPVV_INFOMSG("target_teams_distribute_parallel_for")
  OMPVV_GET_ERRORS(errors_bf)

!$omp target teams distribute parallel do map(from:num_teams, num_threads)
  DO i = 1, ARRAY_SIZE, 1
    !$omp atomic write
    num_teams = omp_get_num_teams()
    num_threads(i) = omp_get_num_threads()
    a(i) = a(i) + b(i) * c(i);
  END DO
```

```
  DO i = 1, ARRAY_SIZE, 1
  OMPVV_TEST_AND_SET(errors_af, (a(i) .NE. (1 + (b(i) * c(i)))))
  IF (num_threads(i) .EQ. 1) THEN
    alert_num_threads = alert_num_threads + 1
  END IF
  END DO

  ! Rise lack of parallelism alerts
  WRITE(msgHelper, *) "Test operated with one team.  &
        &Parallelism of teams distribute can't be guaranteed."
  OMPVV_WARNING_IF(num_teams == 1, msgHelper);

  WRITE(msgHelper, *) "Test operated with one thread &
        &in all the teams. Parallel clause had no effect"
  OMPVV_WARNING_IF(alert_num_threads == ARRAY_SIZE, msgHelper);

  OMPVV_GET_ERRORS(errors_af)
  target_teams_distribute_parallel_for = errors_bf - errors_af
END FUNCTION target_teams_distribute_parallel_for
```

How to Generate Opt-Report Info?

- Use "-qopt-report" flag to emit a YAML format opt-report for the source file being compiled.
  - icpx –fiopenmp –fopenmp-target=spir64 –qopt-report foo.c
  - This generates two files called "icx8iguij.yaml, foo.opt.yaml" containing the opt-report messages
- Use opt-viewer.py (from llvm/tools/opt-viewer) to create html file from the YAML file.
  - Opt-viewer.py foo.opt.yaml
- Use any web-browser to open the html file with the opt-report messages displayed inline with the original souce code
  - Firefox html/foo.c.html

Using clang style opt-reports with “icpx –fiopenmp”

```
#include <stdio.h>

int x = 10;

void foo() {
#pragma omp parallel
  printf("In parallel\n");

#pragma omp target map(tofrom:x)
  {
  #pragma omp atomic
  x += 1;
  #pragma omp flush
  }
}
```

Compilation command

```
$ icx -c -O0 -fiopenmp -fopenmp-targets=spir64 -qopt-report=3 omp_optrpt_example.c
$ ls *yaml
icx8iGUij.opt.yaml  omp_optrpt_example.opt.yaml
```

Since the input file (`omp_optrpt_example.c`) contains a target constructs, and the compilation command uses `-fopenmp-targets=spir64` two different `.opt.yaml` files are generated, one for the target compilation (`omp_optrpt_example.opt.yaml`), and another for the host compilation (`icx8iGUij.opt.yaml`).

Format of opt-report messages in yaml files

```
--- !Missed
Pass:            openmp
Name:            Region
DebugLoc:        { File: omp_optrpt_example.c, Line: 13, Column: 15 }
Function:        foo
Args:
  - Construct:       flush
  - String:          ' unsupported construct ignored by clang'
...
```

```
--- !Passed
Pass:            openmp
Name:            Region
DebugLoc:        { File: omp_optrpt_example.c, Line: 9, Column: 1 }
Function:        __omp_offloading_807_9071537__Z3foo_l9
Args:
  - Construct:       target
  - String:          ' construct transformed'
...
```

Get html files from yaml files

```
$ export PYTHON_VERSION=v3.7.4
$ $ICS_WSDIR/llvm/llvm/tools/opt-viewer/opt-viewer.py omp_optrpt_example.opt.yaml
For faster parsing, you may want to install libYAML for PyYAML
Reading YAML files...
```

View the Host Compilation HTML Report

```
$ $ICS_WSDIR/llvm/llvm/tools/opt-viewer/opt-viewer.py icx8iGUij.opt.yaml
$ firefox html/omp_optrpt_example.c.html&
```

Line	Hotness	Optimization	Source	Inline Context
1			`#include <stdio.h>`	
2				
3			`int x = 10;`	
4				
5			`void foo() {`	
		prologepilog	56 stack bytes in function	foo
		asm-printer	68 instructions in function	foo
6			`#pragma omp parallel`	
		openmp	parallel construct transformed	foo.DIR.OMP.PARALLEL.2
		inline	foo.DIR.OMP.PARALLEL.2 not inlined into foo because it should never be inlined (cost=never): no alwaysinline attribute	foo
		prologepilog	8 stack bytes in function	foo.DIR.OMP.PARALLEL.2
		asm-printer	12 instructions in function	foo.DIR.OMP.PARALLEL.2
7			`  printf("In parallel\n");`	
		inline	printf will not be inlined into foo because its definition is unavailable	foo
		inline	printf will not be inlined into foo.DIR.OMP.PARALLEL.2 because its definition is unavailable	foo.DIR.OMP.PARALLEL.2
8				
9			`#pragma omp target map(tofrom:x)`	
		openmp	target construct transformed	__omp_offloading_807_9071...
		inline	__omp_offloading_807_9071537__Z3foo_l9 not inlined into foo because it should never be inlined (cost=never): no alwaysinline attribute	foo
		sdagisel	FastISel missed	foo
		prologepilog	24 stack bytes in function	__omp_offloading_807_9071...
		asm-printer	16 instructions in function	__omp_offloading_807_9071...
10			`  {`	
11			`  #pragma omp atomic`	
12			`  x += 1;`	
		sdagisel	FastISel missed	__omp_offloading_807_9071...
13			`  #pragma omp flush`	
		openmp	flush construct transformed	foo
14			`  }`	
15			`}`	

View Device Compilation HTML Report

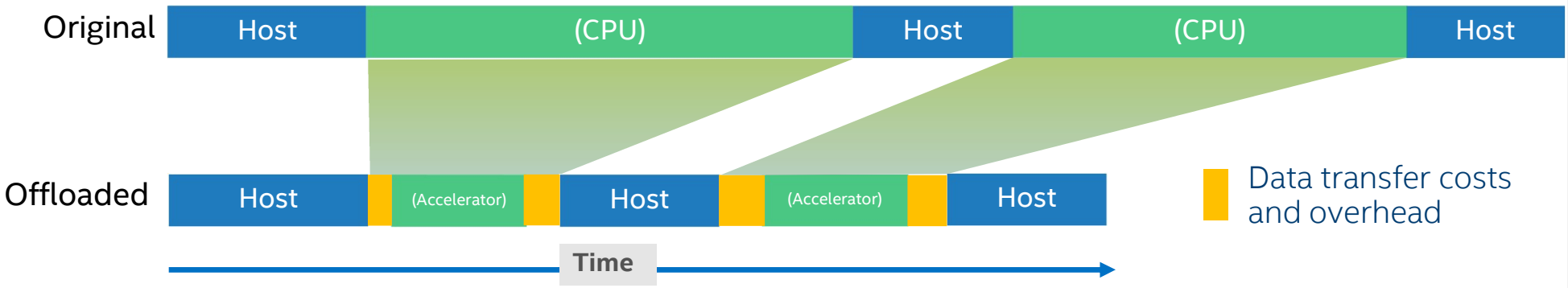
```
$ $ICS_WSDIR/llvm/llvm/tools/opt-viewer/opt-viewer.py omp_optrpt_example.opt.yaml
$ firefox html/omp_optrpt_example.c.html&
```

Line	Hotness	Optimization	Source	Inline Context
1			`#include <stdio.h>`	
2				
3			`int x = 10;`	
4				
5			`void foo() {`	
6			`#pragma omp parallel`	
		openmp	parallel construct ignored	foo
7			`printf("In parallel\n");`	
		inline	printf will not be inlined into foo because its definition is unavailable	foo
8				
9			`#pragma omp target map(tofrom:x)`	
		openmp	target construct transformed	__omp_offloading_807_9071...
10			`{`	
11			`#pragma omp atomic`	
		openmp	atomic update construct handled by clang	foo
12			`x += 1;`	
		inline	__kmpc_atomic_load will not be inlined into foo because its definition is unavailable	foo
		inline	__kmpc_atomic_compare_exchange will not be inlined into foo because its definition is unavailable	foo
		inline	__kmpc_atomic_load will not be inlined into __omp_offloading_807_9071537__Z3foo_l9 because its definition is unavailable	__omp_offloading_807_9071...
		inline	__kmpc_atomic_compare_exchange will not be inlined into __omp_offloading_807_9071537__Z3foo_l9 because its definition is unavailable	__omp_offloading_807_9071...
13			`#pragma omp flush`	
		openmp	flush unsupported construct ignored by clang	foo
14			`}`	
15			`}`	
16				
17				

Summary: 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)



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

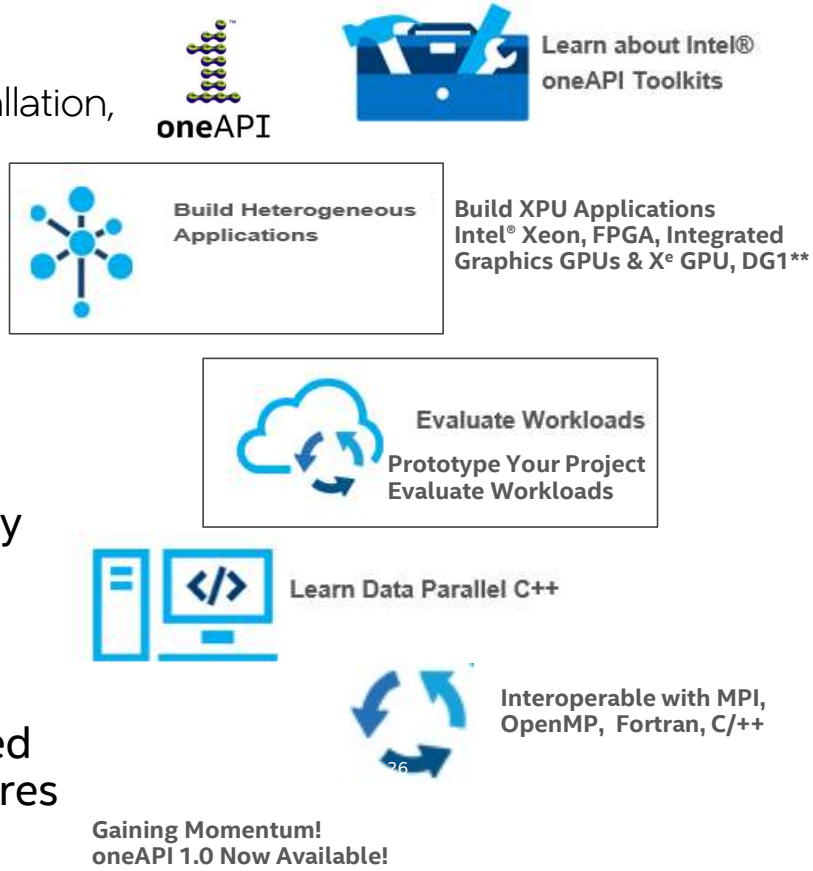
Develop On-Prem – Download & Develop - Get them here - software.intel.com/oneAPI

Community Collaboration

Community – Working with community closely for OpenMP 5.2/6.0 support.

OpenMP Specification

Stay with Standard Body - Cross-industry, open, standards-based unified programming model across architectures – Learn more here – openmp.org



Gaining Momentum!
oneAPI 1.0 Now Available!

Additional Resources

- Intel® oneAPI Base Toolkit
- Intel® oneAPI Base & HPC Toolkit
- Intel® oneAPI Base & IoT Toolkit
- Intel® oneAPI Base & Rendering Toolkit

-
- Replay will be available here at the end of the week: https://techdecoded.intel.io/essentials/#gs.1t92xk
- Other webinars for you: https://techdecoded.intel.io/webinar-registration/upcoming-webinars/#gs.1t90np

intel®