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.

- 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 © 2019, 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.

### 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
Agenda

- Xe GPU: Recap
- The Big Picture of oneAPI Software Stack
- OpenMP Offloading Model and Subset for Xe GPUs
- JIT Compilation Flow
- OpenMP Offload Examples
- Case Studies: NWChem and Mini-Weather
- Summary
Multi-tile GPU  By David Blythe at HotChips’2020

1-Tile  >10 FP32 TFLOPS
2-Tile  >20 FP32 TFLOPS
4-Tile  >40 FP32 TFLOPS
The Big Picture with Frameworks & HPC/AI Applications

HPC/AI C/C++ and Fortran Optimized Applications

Portability frameworks: Kokkos, RAJA, etc.

Direct Programming

- Data Parallel C++
- OpenMP C/C++
- OpenMP FORTRAN
- Future languages

API-based Programming

- AI/Math
- Parallel
- Visual

  - oneDNN
  - oneTBB
  - oneVPL

  - oneMKL
  - oneDPL
  - Media SDK

  - oneCCL
  - MPI

  - oneDAL
  - MPI

Analysis tools: Debuggers, Profilers, Advisors

OS, or GPU driver, OpenCL RT, low-level runtime, etc.

CPU

GPU / Xe Accelerator

AI / FPGA
OpenMP Offload and Subset for Xe GPUs
Target Teams Construct

- Creates multiple teams with a leading thread in each team, effectively creates a set of thread teams (league)
- Synchronization does not apply across teams.
OpenMP 4.5 and 5.0/5.1 Subset for Xe Accelerator

- **Offload code to run on a target device**
  - `omp target` [clause[[], clause],...]
    - structured-block
  - `omp declare target`
    - [function-definitions-or-declarations]

- **Map variables to a target device**
  - `map` ([map-type:] list) // map clause
    - map-type := alloc | tofrom | to | from
  - `omp target [enter | exit] data` [clause[, clause],...]
    - structured-block
  - `omp target update` [clause[, clause],...]
  - `omp declare target`
    - [variable-definitions-or-declarations]
  - `omp declare mapper`

- **Work-sharing for acceleration**
  - `omp teams/single/master` [clause[, clause],...]
  - `omp distribute` [for | do] [clause[, clause],...]
    for-loops

- **Parallel and simd code to run on GPU**
  - `omp parallel for/do [simd]` [clause[, clause],...]
  - `omp simd`
    - A set of composite and combined constructs [clause[, clause],...]
      - E.g. `#pragma omp target teams distribute parallel for simd`

- **Synchronization**
  - `omp atomic` [clause[, clause],...]
    - map-type := alloc | tofrom | to | from
  - `omp critical` [clause[, clause],...]
    - structured-block
OpenMP Runtime APIs

Runtime API list for offline reference

Runtime support routines on CPU Host

- EXTERN int omp_get_num_devices(void);
- EXTERN int omp_get_initial_device(void);
- EXTERN void *omp_target_alloc(size_t size, int device_num);
- EXTERN void omp_target_free(void *device_ptr, int device_num);
- EXTERN int omp_target_is_present(void *ptr, int device_num);
- EXTERN void *omp_target_alloc(size_t size, int device_num);
- EXTERN void omp_target_free(void *device_ptr, int device_num);
- EXTERN int omp_target_memcpy(void *dst, void *src, size_t length, size_t
  dst_offset, size_t src_offset, int dst_device, int src_device);
- EXTERN int omp_target_memcpy_rect(void *dst, void *src, size_t element_size,
  int num_dims, const size_t *volume, const size_t *dst_offsets,
  const size_t *src_offsets, const size_t *dst_dimensions,
  const size_t *src_dimensions, int dst_device, int src_device);
- EXTERN int omp_target_associate_ptr(void *host_ptr, void *device_ptr, size_t
  size, size_t device_offset, int device_num);
- EXTERN int omp_target_disassociate_ptr(void *host_ptr, int device_num);
- EXTERN int omp_is_initial_device(void);
- EXTERN int omp_get_initial_device(void);
- EXTERN void kmp_global_barrier_init(void); // Intel extension
- EXTERN void kmp_global_barrier(void); // Intel extension
- EXTERN void omp_set_default_device(int dev_num);
- EXTERN int omp_get_default_device(void);

Device Runtime Routines for GPU

- EXTERN int omp_get_team_num(void);
- EXTERN int omp_get_num_teams(void);
- EXTERN int omp_get_team_size(int);
- EXTERN int omp_get_thread_num(void);
- EXTERN int omp_get_num_threads(void);
- EXTERN int omp_in_parallel(void);
- EXTERN int omp_get_max_threads(void);
- EXTERN int omp_get_device_num(void);
- EXTERN int omp_get_num_devices(void);
JIT Compilation + Execution Flow
Just-In-Time (JIT) Compilation Flow

Compiler driver → CLANG FE → IFX FE → LLVM-IR opt's → objgen

LLVM-IR "Byte Code"

Host-path

Compiler driver → CLANG FE → IFX FE → LLVM-IR opt's → objgen → obj

HW specific Code-gen

Compiler driver → CLANG FE → IFX FE → LLVM-IR opt's → objgen → .bc

LLVM-IR linker → LLVM-IR -to- SPRIV

.obj · .spv → SPRIV -to- .obj offload-wrapper

.JIT compiler

.runtime

.device-code

.FAT binary

host-code

host-linker

.exe

某个

JIT compiler

SPRIV -to- LLVM-IR opt's

HW specific Code-gen

Device-path

Host: regular IA code
 Device: Single SPIR-V wrapped in .exe 'data' section
OpenMP Offloading Examples
Target Region Example: saxpy

```c
void saxpy() {
    float a, x[ARRAY_SZ], y[ARRAY_SZ];
    double t = 0.0;
    double tb, te;
    tb = omp_get_wtime();
    #pragma omp target teams distribute parallel for
    for (int i = 0; i < ARRAY_SZ; i++) {
        y[i] = a * x[i] + y[i];
    }
    te = omp_get_wtime();
    t = te - tb;
    printf("Time of kernel: %lf\n", t);
}
```

```
icx -fiopenmp -fopenmp-targets=spir64 -o saxpy saxpy.c
```
Calling Functions Inside Target Area

```c
#pragma omp declare target

int devicefunc(){
...
}
#pragma omp end declare target

#pragma omp target
{
    result = devicefunc();
}
```

```c
subroutine devicefunc()

!$omp declare target device_type(device)
...
end subroutine

program main

!$omp target
    call devicefunc()
!$omp end target
end program
```
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
DPC++ and OpenMP Composability

```cpp
#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;
        });
    });
}

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
Case Study: NWChem
TCE CCSD (T)
TCE: Tensor Contraction Engine
CCSD(T): Coupled Cluster w/ Singles Doubles, Triples approximation
NWChem Application

- Computational chemistry software package
  - Quantum chemistry
  - Molecular dynamics
- Designed for large-scale supercomputers
- Developed at the EMSL at PNNL
  - EMSL: Environmental Molecular Sciences Laboratory
  - PNNL: Pacific Northern National Lab
- URL: http://www.nwchem-sw.org
All kernels expose the same structure
7 perfectly nested loops
Some kernels contain inner product loop (then, 6 perfectly nested loops)
Trip count per loop is equal to “tile size” (20-30 in production)

Naïve data allocation (tile size 24)
- Per-array transfer for each target construct
  - triplesx: 1458 MB
  - t2sub, v2sub: 2.5 MB

Example Kernel (1 of 27 in total)

```c
subroutine offl_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
 h7d,triplesx,t2sub,v2sub)
!
 !$omp target
double precision triplesx(h3d*h2d,h1d,p6d,p5d,p4d)
double precision t2sub(h7d,p4d,p5d,h1d)
double precision v2sub(h3d*h2d,p6d,h7d)
!$omp end target
do p4=1,p4d
do p5=1,p5d
do p6=1,p6d
do h1=1,h1d
do h7=1,h7d
do h2h3=1,h3d*h2d
  triplesx(h2h3,h1,p6,p5,p4)=triplesx(h2h3,h1,p6,p5,p4)
  - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
end do
doub
e end do
e end
doub
e end do
de
!$omp end teams distribute parallel do
e!$omp end target
doub
e end subroutine
```

1.5GB data transferred (host to device)
1.5GB data transferred (device to host)

1.5GB data transferred (host to device)
Invoking the Kernels / Data Management

- Simplified pseudo-code of the actual NWChem TCE CCSD(T) code

```c
!$omp target enter data alloc(triplesx(1:tr_size))
c    for all tiles
    do ... 
c      call zero_triplesx(triplesx)
c      do ... 
c      call comm_and_sort(t2sub, v2sub)
!$omp target data map(to:t2sub(t2_size)) map(to:v2sub(v2_size))
    if (...) 
      call sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,h7,triplesx,t2sub,v2sub)
    end if 
c    same for sd_t_d1_2 until sd_t_d1_9
!$omp target end data
    end do 
    do ... 
c    Similar structure for sd_t_d2_1 until sd_t_d2_9, incl. target data 
    end do 
    call sum_energy(energy, triplesx)
    end do
!$omp target exit data release(triplesx(1:size))
```

- Reduced data transfers:
  - **triplesx**:
    - allocated once
    - always kept on the target
  - **t2sub, v2sub**:
    - allocated after comm.
    - kept for (multiple) kernel invocations

Allocate 1.5GB data once, stays on device.

Update 4MB of data for (potentially) multiple kernels.
Mini-Weather Code
OpenMP Offload Example: miniWeather

https://github.com/mrnorman/miniWeather

miniWeather is a mini-app developed by Dr. Matthew Norman (OLCF). The code mimics the basic dynamics seen in atmospheric weather and climate:

- Solves 2D Euler equations of compressible flow
- 4th-order central finite-volume scheme
- 3rd-order Runge-Kutta time integration
- 4th-order hyperviscosity

Enabled for multiple parallel programming models: MPI, OpenMP, OpenMP offload, OpenACC, C++ (w/ CUDA, HIP in backend)

Example of mapping variables between the host and target device

```c
#pragma omp target data
map(to:state_tmp[:](nz+2*hs)*(nx+2*hs)*NUM_VARS],hy_dens_cell[:nz+2*hs],hy_dens_theta_cell[:nz+2*hs],hy_dens_int[:nz+1],hy_dens_theta_int[:nz+1],hy_pressure_int[:nz+1]) \
map(alloc:flux[:](nz+1)*(nx+1)*NUM_VARS],tend[:](nz nx NUM_VARS],sendbuf_l[:hs*num NUM_VARS] sendbuf_r[:hs*num NUM_VARS],recvbuf_l[:hs*num NUM_VARS],recvbuf_r[:hs*num NUM_VARS]) \
map(tofrom:state[:](nz+2*hs)*(nx+2*hs)*NUM_VARS])
```

Example of the target teams construct

```c
#pragma omp target teams distribute parallel for simd collapse(3)
for (ll=0; ll<NUM_VARS; ll++) {
  for (k=0; k<nz; k++) {
    for (i=0; i<nx; i++) {
      inds = ll*(nz+2*hs)*(nx+2*hs) + (k+hs)*(nx+2*hs) + i+hs;
      indt = ll*nz*nx + k*nx + i;
      state_out[inds] = state_init[inds] + dt * tend[indt];
    }
  }
}
```
miniWeather: Build and run with Intel oneAPI

Compile miniWeather with Intel oneAPI (example for cmake)

```bash
#!/bin/bash
git clone git@github.com:mrnorman/miniWeather.git
cd miniWeather/c/build

source /opt/intel/oneapi/setvars.sh
source /opt/intel/oneapi/Gen9_setup_beta09_20.04LTS.sh

./cmake_clean.sh

export I_MPI_CC=icx
export I_MPI_CXX=icpx
export I_MPI_F90=ifx

./openmp45
```

Example of running miniWeather app on Intel Gen9 Graphics

```bash
$ source /opt/intel/oneapi/setvars.sh
$ clinfo -l
Platform #0: Intel(R) OpenCL
   `-- Device #0: Intel(R) Core(TM) i7-6770HQ CPU @ 2.60GHz
Platform #1: Intel(R) OpenCL HD Graphics
   `-- Device #0: Intel(R) Gen9 HD Graphics NEO
Platform #2: Intel(R) OpenCL HD Graphics
   `-- Device #0: Intel(R) Gen9 HD Graphics NEO

./openmp45
nx_glob, nz_glob: 2000 1000
dx,dz: 10.000000 10.000000
dt: 0.033333

*** OUTPUT ***
Elapsed Time: 0.000000 / 700.000000
Elapsed Time: 0.033333 / 700.000000

CPU Time: 22.8183 sec
d_mass: 1.562607e-14
d_te: -4.914851e-06
```

Output data passed correctness check
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)