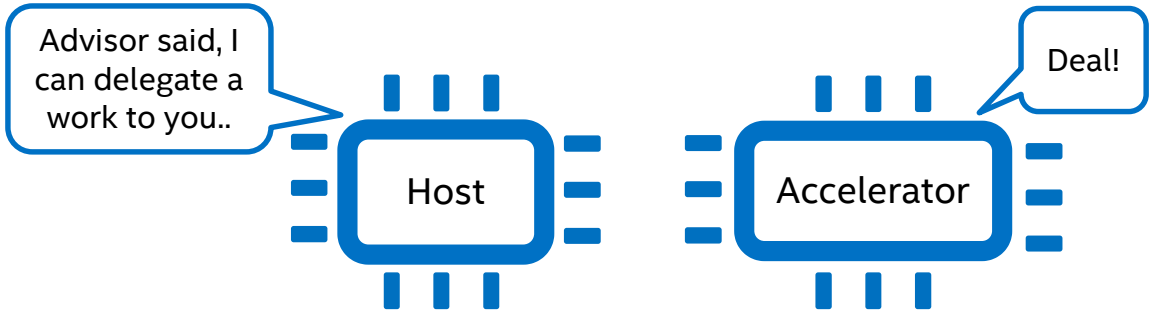
OFFLOAD ADVISOR

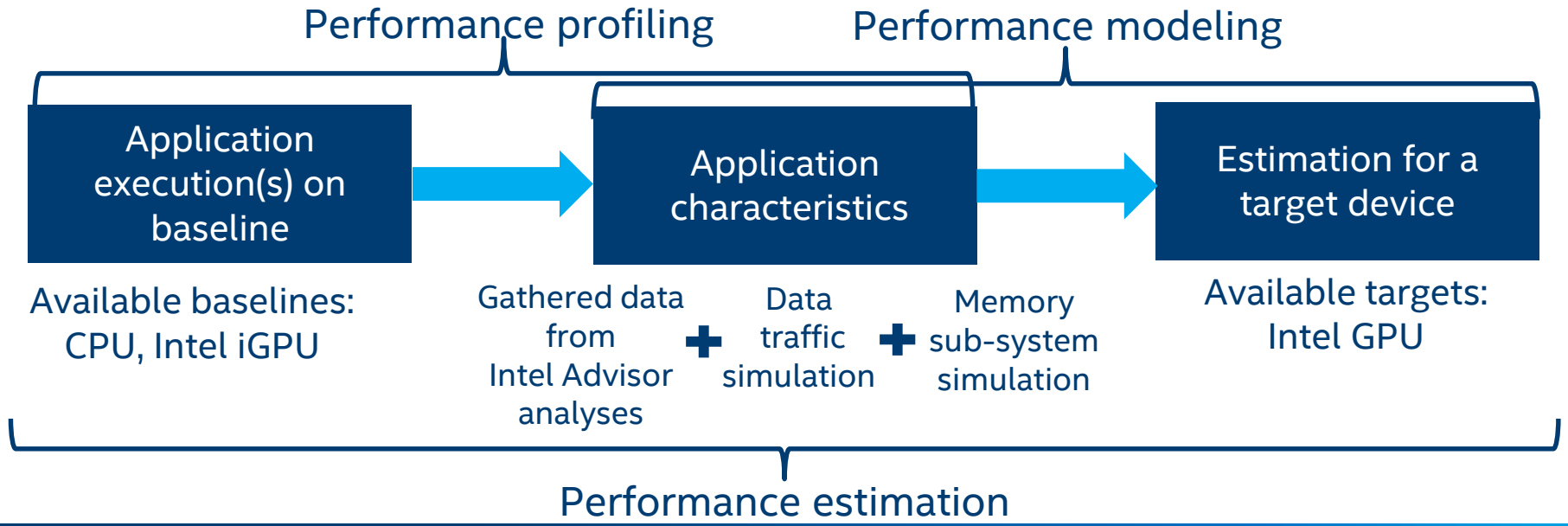
Purpose of Offload Advisor

- Offload Advisor is designed to help users to port their codes to accelerators
  - It can identify the portions of a code that are profitable to be offloaded to an accelerator (e.g. GPU)
  - It can also predict the code's performance if run on an accelerator and lets you experiment with accelerator configuration parameters

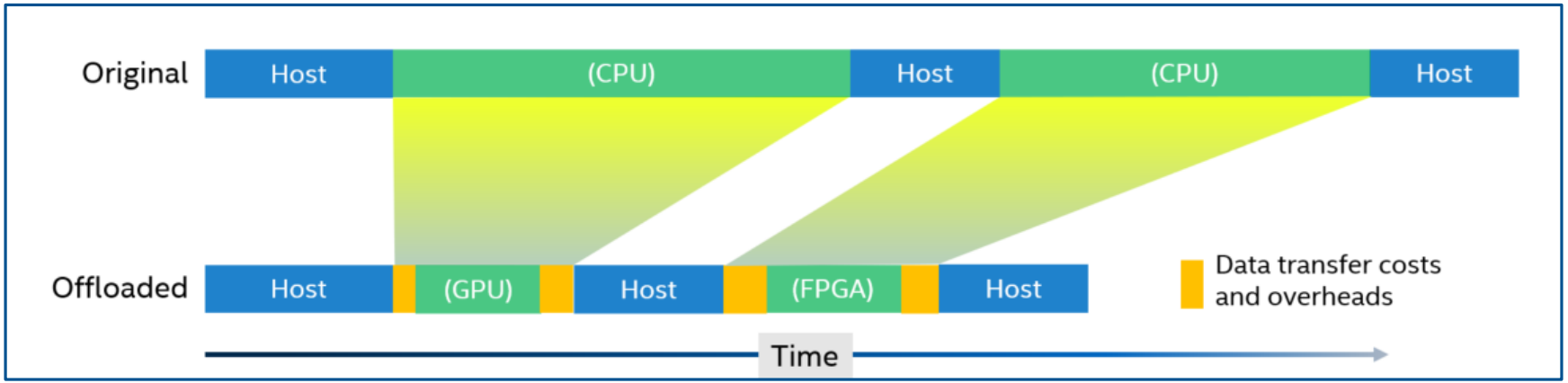


How it works

- Offload Advisor reuses Intel Advisor powerful characterization framework
- Also, it is enriched with data traffic, memory sub-system simulation and analytical performance modeling to enable new Offload Advisor workflow



How it works (CPU to GPU offloading)



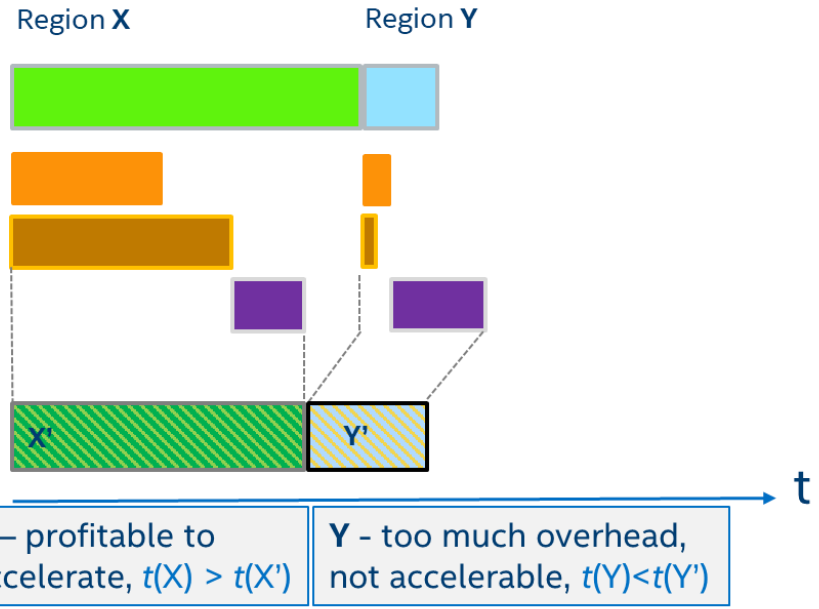
- Data transfer taxes: copy data between CPU and GPU
- Offload taxes: time to place task to GPU task dispatcher

How it works: Region time calculation

Execution time on baseline platform (CPU)

- Execution time on accelerator. Estimate assuming bound exclusively by Compute
- Execution time on accelerator. Estimate assuming bound exclusively by caches/memory
- Offload Tax estimate (data transfer + invoke)

Final estimated time on target device (GPU)



$$t_{region} = \max(t_{compute}, t_{memory\ subsystem}) + t_{data\ transfer\ tax} + t_{invocation\ tax}$$

How it works: Calculation of total time for a loop hierarchy

We minimize the total time spent in a loop hierarchy by varying offload strategies U (offload/non-offload, #threads for each component $loop_i$ of loop nest)

Objective function :

$$\boldsymbol{T_{all}} = \min_{U=\{uf_1, uf_2, ..\}} \left(\sum_i \boldsymbol{T_i} + t_{data\ transfer} + t_{invoke} + T_{cpu} \right)$$

Reject loop nests for which $\boldsymbol{T}$ (x86) / $\boldsymbol{T_{all}}$ (x86+"X") < 1.0

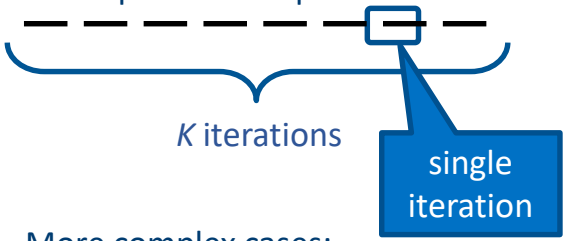
$$\boldsymbol{T_i} = max \begin{cases} T_i^{Comp\_only} \\ T_i^{M_k\_only}\left(M_i^k\right) = \dfrac{\boldsymbol{M_i^k}}{\boldsymbol{BW_k}} \end{cases}$$

This is effective "balance" (throughput) model

Under algorithmic constraints (Dependencies and TripCount/Granularity)

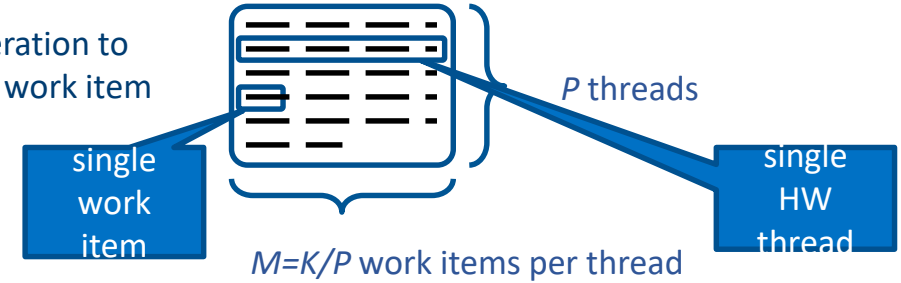
How it works: GPU Execution Time Model

baseline: loop with *N* iterations
no loop-carried dependencies



Model single iteration to single work item

target GPU: kernel with ***K* work items** executed by *P* HW threads



More complex cases:
- parallel threads
- vectorized loop

estimate ***K*: user iteration space**

P limited with HW capabilities:
- number of EU
- number of ALU per EU
- SIMD width of ALU in bytes

$$P = EU\_count * ALU\_per\_EU * ALU\_SIMD\_width$$

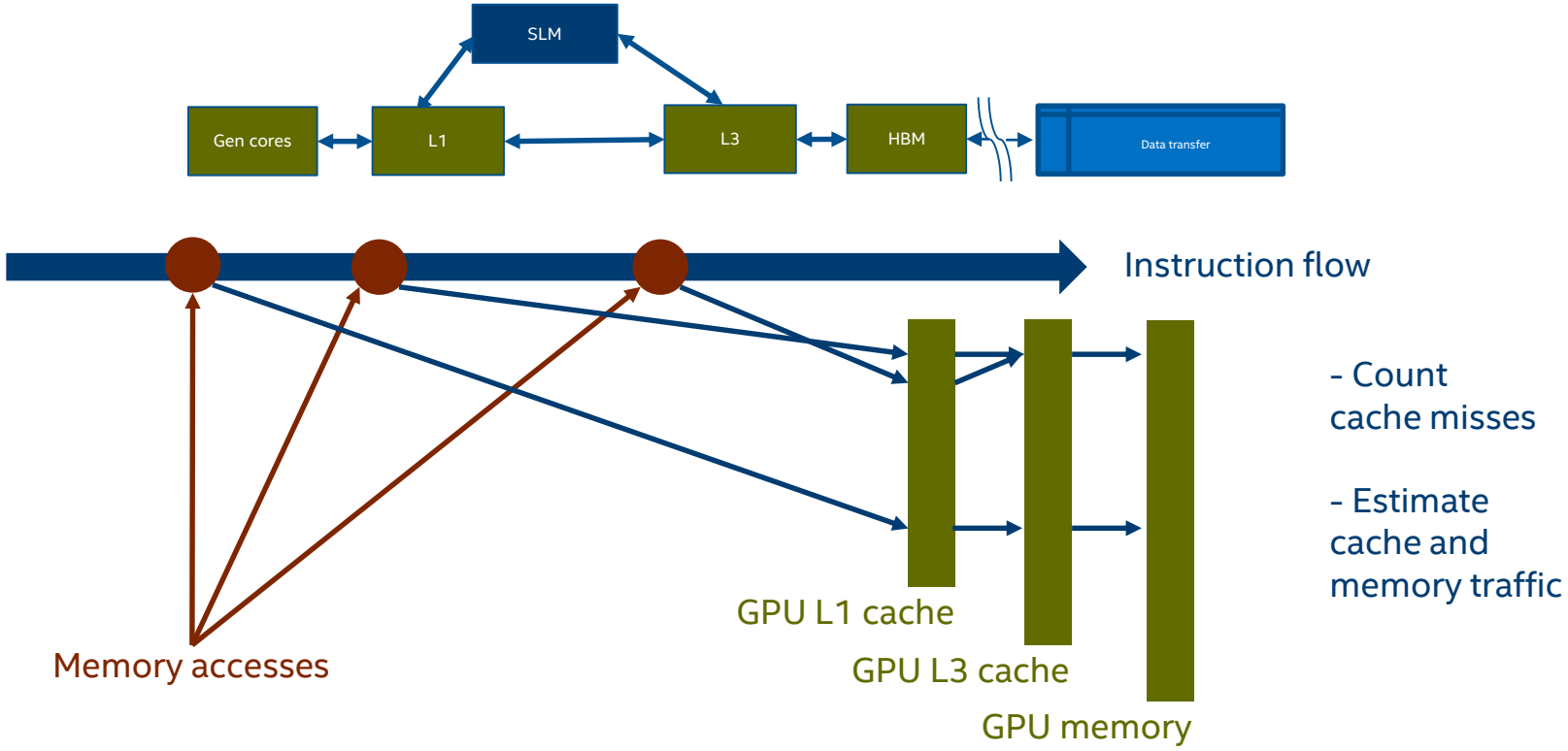
$$M = ceil(K/P)$$

$$T = M * \frac{\sum_i N_i * C_i * VL_{baseline}}{Freq_{GPU}}$$

cost of single loop iteration (in GPU cycles)

C_i - latency of instruction i on the target platform (GPU)
N_i - number of instructions i on the baseline platform (x86)
$VL_{baseline}$ - vector length on the baseline platform (x86)

How it works: GPU Cache & Memory Traffic Model



- Count cache misses

- Estimate cache and memory traffic

LAB ACTIVITIES

The Lab Activities

Activity 0: Making the Project on GitLab

Activity 1: Building and running N-body base version

Activity 2: Running performance estimation for the base version

Activity 3: Looking at the estimation results

Activity 4: Rewriting code using DPC++

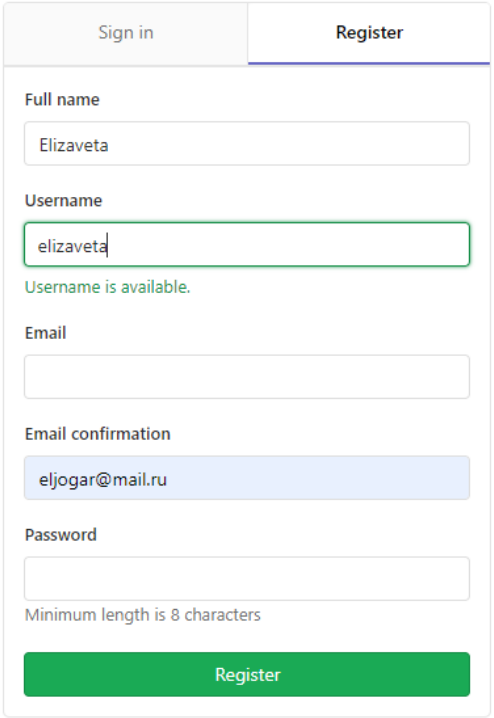
Activity 5: Building N-body DPC++ version

Activity 6: Comparing base and DPC++ versions

ACTIVITY 0: MAKING THE PROJECT

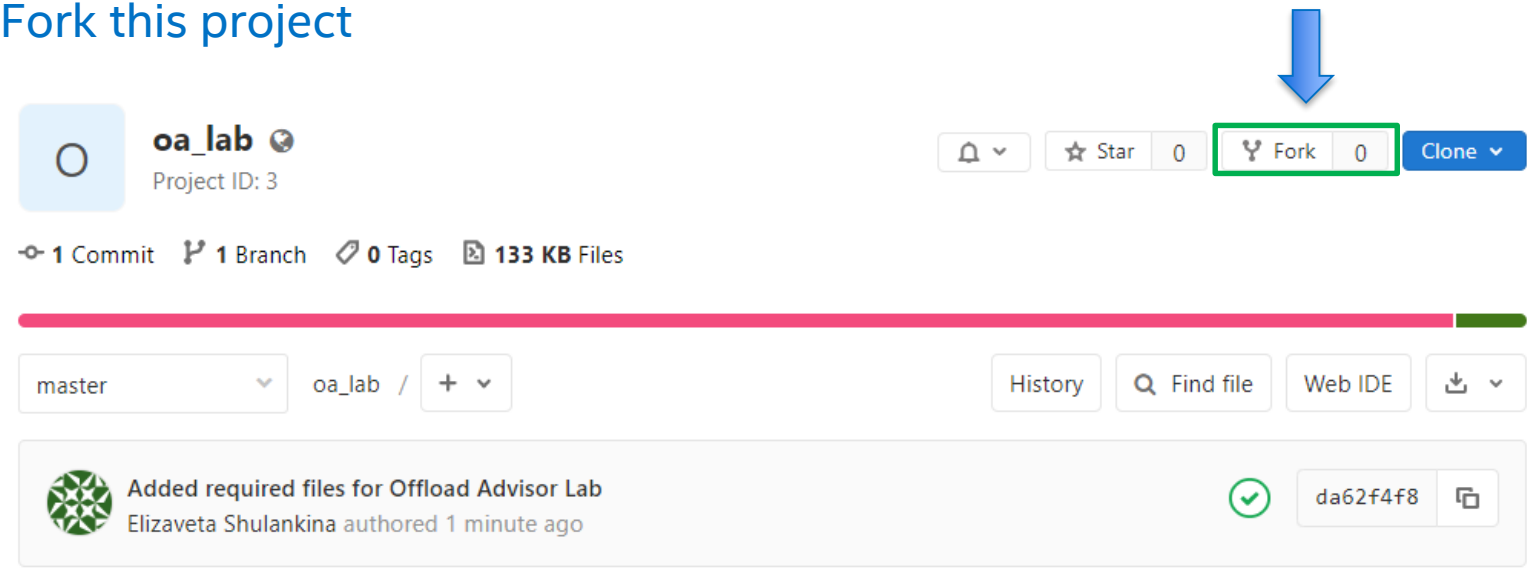
Activity 0: Register

1. Navigate to https://oneapi.team
2. Register



Activity 0: Fork a Project

1. Navigate to https://oneapi.team/dsivkov/offload_advisor
2. Fork this project



ACTIVITY 1: BUILDING N-BODY BASE VERSION

Activity 1: Build & Run

Create your own pipeline to build & run the N-Body base version:

- Add the following jobs to **.gitlab-ci.yml** in your project:

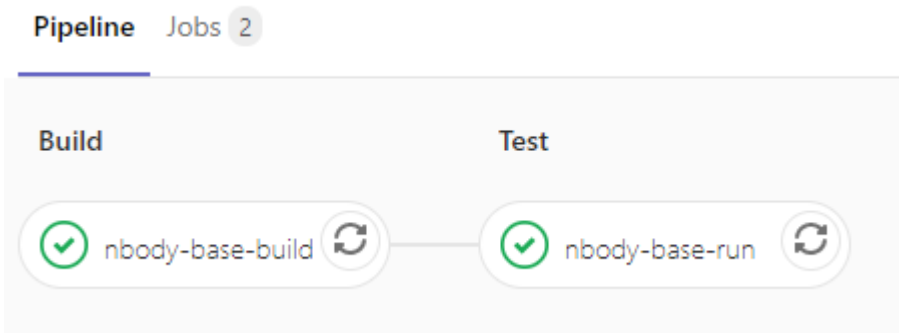
Link: nbody-base-build

```
nbody-base-build:
  stage: build
  tags:
    - oneapi
  script:
    - make -C ./nbody/base
  artifacts:
    paths:
      - ./nbody/base/nbody
```

Link: nbody-base-run

```
nbody-base-run:
  stage: test
  tags:
    - oneapi
  script:
    - ./nbody/base/nbody $NBODY_ARGS
  dependencies:
    - nbody-base-build
```

Activity 1: Check pipeline status




```
$ ./nbody/base/nbody $NBODY_ARGS
================================
 Initialize Gravity Simulation
 nPart = 4000; nSteps = 1000; dt = 0.1
----------------------------------------
 s       dt      kenergy     time (s)
----------------------------------------
 50      5       66.863      0.67641
 100     10      369.19      0.58348
 150     15      1089.7      0.58435
 200     20      3005.8      0.58221
 250     25      15704       0.5829
 300     30      10062       0.57951
 350     35      6497.2      0.57881
 400     40      5674        0.58026
 450     45      5350.2      0.57886
 500     50      5019.5      0.58176
 550     55      5043.5      0.57916
 600     60      4815.3      0.57909
 650     65      5010.3      0.5809
 700     70      4783        0.57874
 750     75      4797.8      0.58029
 800     80      4987.9      0.57866
 850     85      4699.8      0.58215
 900     90      4534        0.57835
 950     95      4907.4      0.58682
 1000    100     4915.7      0.58003
# Total Time (s)     : 11.713
================================
```

ACTIVITY 2: RUNNING PERFORMANCE ESTIMATION FOR THE BASE VERSION

How to run Performance Estimation

There are three methods varying in simplicity and flexibility to run performance profiling and performance modeling. Performance profiling and performance modeling are used together to derive performance estimates.

Commands	Notes
Stand-alone **run_oa.py** script	Most simple and least flexible. **Does not support MPI applications.**
Combination of **collect.py** and **analyze.py** scripts	Somewhat simple and flexible. **Does not support MPI applications.**
Combination of Advisor Command Line **advixe-cl** and **analyze.py** script	Least simple and most flexible. **Applicable to MPI applications.**

run_oa.py script (most simple)

This is the most simple method.

It automates the process of invoking performance profiling with reasonable pre-defined options, then runs performance modeling on the resulting profiles to generate performance estimates.

- `python $APM/run_oa.py <advisor_results_dir> -o <apm_results_dir> [options] -- <app_binary> [app_options]`

collect.py & analyze.py (simple and flexible)

This is a middle-of-the-road method that is moderately simple and flexible. The collect.py script automates the process of performance profiling and the analyze.py script implements performance modeling.

- `python $APM/collect.py <advisor_results_dir> [options] -- <app_binary> [app_options]`
- `python $APM/analyze.py <advisor_results_dir> -o <apm_results_dir>`

advixe-cl & analyze.py (most flexible)

This is the most flexible method including calling **advixe-cl** (directly invoking the run of Advisor analyses) and a script **analyze.py** that implements performance modeling.

For each required Advisor analysis, you should run **advixe-cl** with the appropriate parameters. After all the information is collected, you should run **analyze.py**.

Example of Survey analysis run:

- `advixe-cl --collect=survey --auto-finalize --stackwalk-mode=online -static-instruction-mix –-project-dir=<advisor_results_dir> -- <app_binary> [app_options]`

`--dry-run` option for the most flexible method

This parameter helps users to get all advixe-cl commands with parameters that are used when running collect.py.

If we run: **python collect.py** –config gen9 –-dry-run ./test -- ./app

We get the following output:

1. **advixe-cl** -collect **survey** -project-dir ./test -return-app-exitcode -auto-finalize -static-instruction-mix <other predefined options> -- ./app
2. **advixe-cl** -collect **tripcounts** -project-dir ./test -return-app-exitcode -flop -auto-finalize <other predefined options> -- ./app
3. **python collect.py** ./test -a gen -m generic
4. **advixe-cl** -collect **dependencies** -project-dir ./test -return-app-exitcode -<other predefined options> -- ./app

Activity 2

Add a job to run the performance estimation for the N-Body base version:

Link: nbody-base-profile

```
nbody-base-profile:
  stage: profile
  tags:
    - oneapi
  before_script:
    - . . .
  script:
    - mkdir adv_prj

    # Running Offload Advisor
    - advixe-python $APM/collect.py --config=gen9 ./adv_prj -c basic -- ./nbody/base/nbody $NBODY_ARGS
    - advixe-python $APM/analyze.py --config=gen9 --set-parallel=GSimulation.cpp:103,GSimulation.cpp:129 ./adv_prj

    # Running base version of nbody sample
    - . . .
    - ./nbody/base/nbody $NBODY_ARGS | tee -a sample_run.log
  . . .
```

What is the purpose of the red parameters?

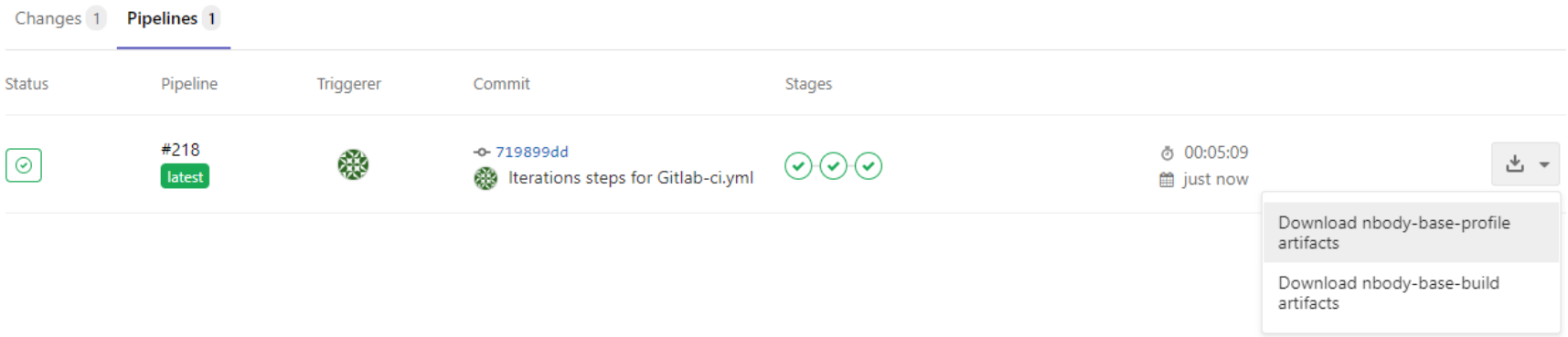
For a demonstration, we reduce the profiling time using the command line parameter:

`-c basic`	Collect basic data: Survey and Tripcounts & FLOPs (No MAP and Dependencies run)

If Offload Advisor doesn't know information about data dependencies for a loop, it will use a pessimistic approach, assuming that they exist. We can give a hint to the Advisor using the following parameter:

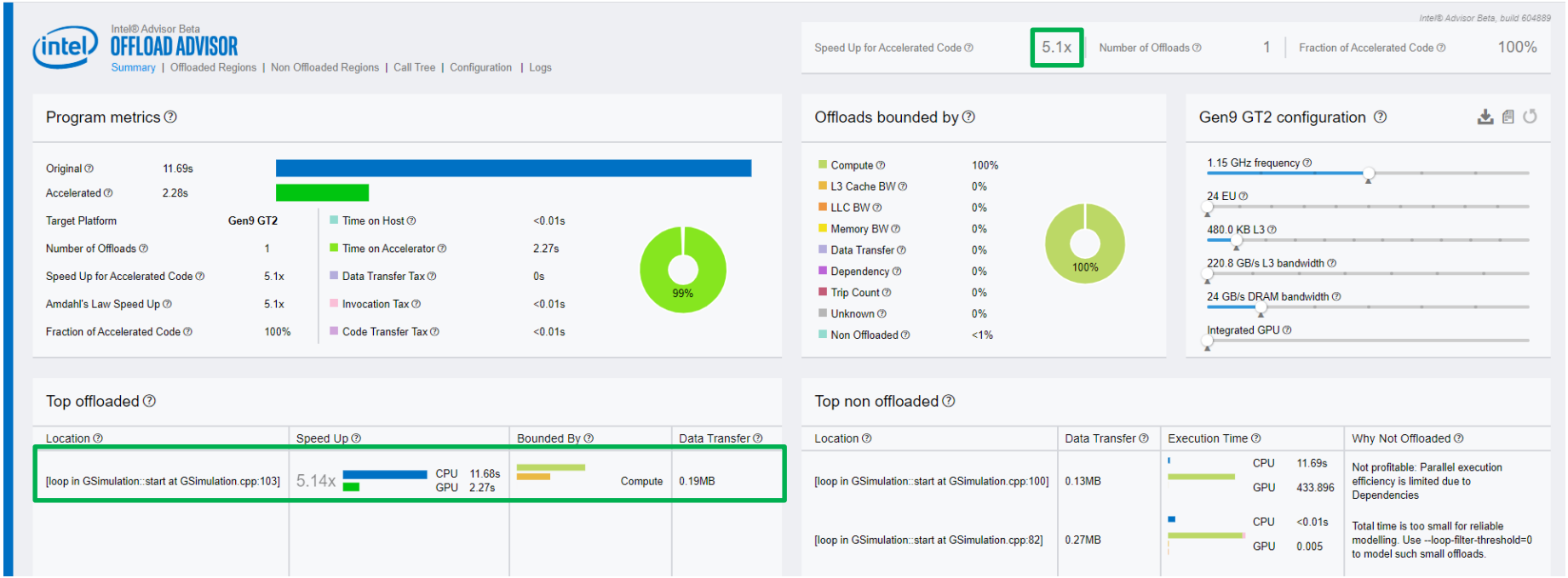
`--set-parallel`	This option allows users to mark specific regions as regions that do not have data dependencies and can be parallel

Activity 2: Download artifacts

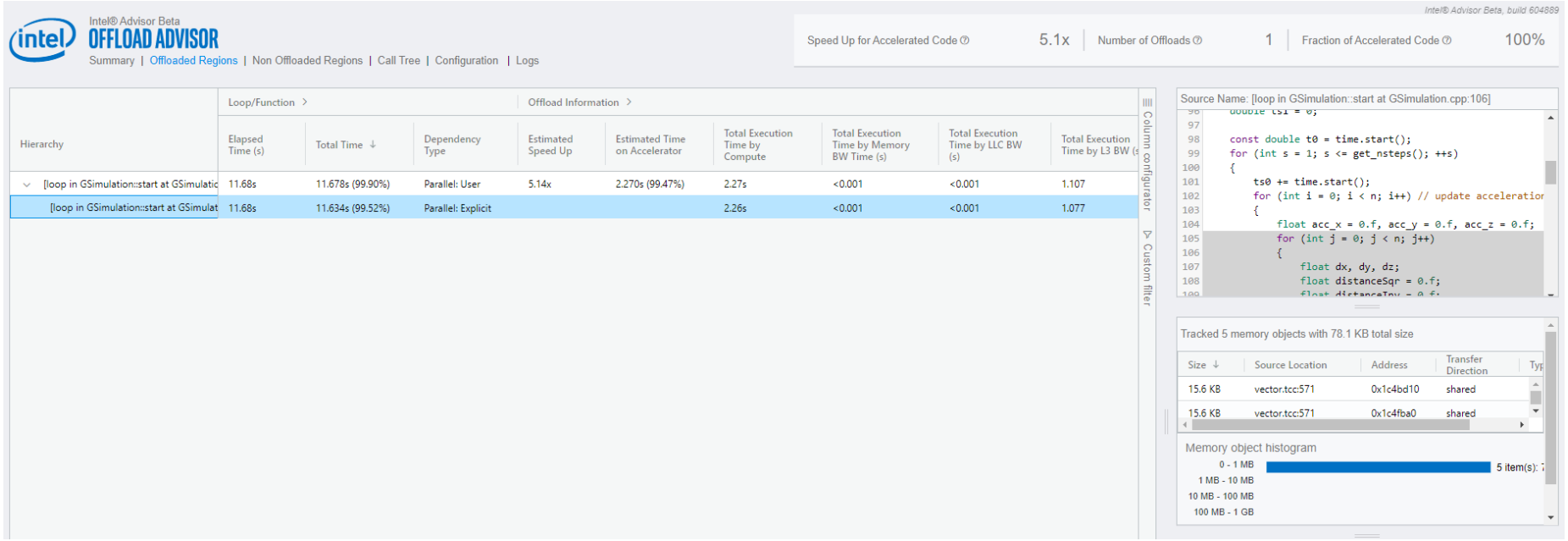


ACTIVITY 3: LOOKING AT THE ESTIMATION RESULTS

Activity 3



Activity 3



ACTIVITY 4: REWRITING CODE USING DPC++

Activity 4

```
const double t0 = time.start();
for (int s = 1; s <= get_nsteps(); ++s)
{
    ts0 += time.start();
    for (int i = 0; i < n; i++) // update acceleration
    {
        float acc_x = 0.f, acc_y = 0.f, acc_z = 0.f;
        for (int j = 0; j < n; j++)
        {
            . . .
```

```
            dx = particles.pos_x[j] - particles.pos_x[i];
            dy = particles.pos_y[j] - particles.pos_y[i];
            dz = particles.pos_z[j] - particles.pos_z[i];

            distanceSqr = dx * dx + dy * dy + dz * dz + softeningSquared;
            distanceInv = 1.0f / sqrtf(distanceSqr);

            acc_x += dx * G * particles.mass[j] * distanceInv * distanceInv * distanceInv;
            acc_y += dy * G * particles.mass[j] * distanceInv * distanceInv * distanceInv;
            acc_z += dz * G * particles.mass[j] * distanceInv * distanceInv * distanceInv;
        }
        particles.acc_x[i] = acc_x;
        particles.acc_y[i] = acc_y;
        particles.acc_z[i] = acc_z;
    }
    energy = 0.f;
```

```
sycl::default_selector selector;
sycl::queue queue(selector, exceptionHandler);

sycl::buffer<float, 1> posX(particles.pos_x.data(), sycl::range<1>(n));
sycl::buffer<float, 1> posY(particles.pos_y.data(), sycl::range<1>(n));
sycl::buffer<float, 1> posZ(particles.pos_z.data(), sycl::range<1>(n));
sycl::buffer<float, 1> velX(particles.vel_x.data(), sycl::range<1>(n));
. . .
sycl::buffer<float, 1> mass(particles.mass.data(), sycl::range<1>(n));
sycl::buffer<float, 1> energy(_energy, sycl::range<1>(n));

const double t0 = time.start();
for (int s = 1; s <= get_nsteps(); ++s)
{
    ts0 += time.start();
    queue.submit([&](sycl::handler &cgh) {
        auto posXBuff = posX.get_access<sycl::access::mode::read>(cgh);
        auto posYBuff = posY.get_access<sycl::access::mode::read>(cgh);
        auto posZBuff = posZ.get_access<sycl::access::mode::read>(cgh);
        auto massBuff = mass.get_access<sycl::access::mode::read>(cgh);
        auto accXBuff = accX.get_access<sycl::access::mode::write>(cgh);
        auto accYBuff = accY.get_access<sycl::access::mode::write>(cgh);
        auto accZBuff = accZ.get_access<sycl::access::mode::write>(cgh);
            cgh.parallel_for<class kernel1>(sycl::range<1>(n), [=](sycl::id<1> index) {
                int i = index.get(0);
                float acc_x = 0.f, acc_y = 0.f, acc_z = 0.f;
                for (int j = 0; j < n; j++)
                {
                    . . .

                    dx = posXBuff[j] - posXBuff[i];
                    dy = posYBuff[j] - posYBuff[i];
                    dz = posZBuff[j] - posZBuff[i];

                    distanceSqr = dx * dx + dy * dy + dz * dz + softeningSquared;
                    distanceInv = 1.0f / sycl::sqrt(distanceSqr);

                    acc_x += dx * G * massBuff[j] * distanceInv * distanceInv * distanceInv;
                    acc_y += dy * G * massBuff[j] * distanceInv * distanceInv * distanceInv;
                    acc_z += dz * G * massBuff[j] * distanceInv * distanceInv * distanceInv;
                }
                accXBuff[i] = acc_x;
                accYBuff[i] = acc_y;
                accZBuff[i] = acc_z;
            });
    }).wait_and_throw();
```

ACTIVITY 5: BUILDING N-BODY DPC++ VERSION

Activity 5

- Disable **nbody-base-run** and **nbody-base-profile** jobs (comment them)
- Add the following jobs to **.gitlab-ci.yml** in your project:

Link: nbody-dpcpp-build

```
nbody-dpcpp-build:
  stage: build
  tags:
    - oneapi
  script:
    - make -C ./nbody/dpcpp
  artifacts:
    paths:
      - ./nbody/dpcpp/nbody
```

Link: nbody-run

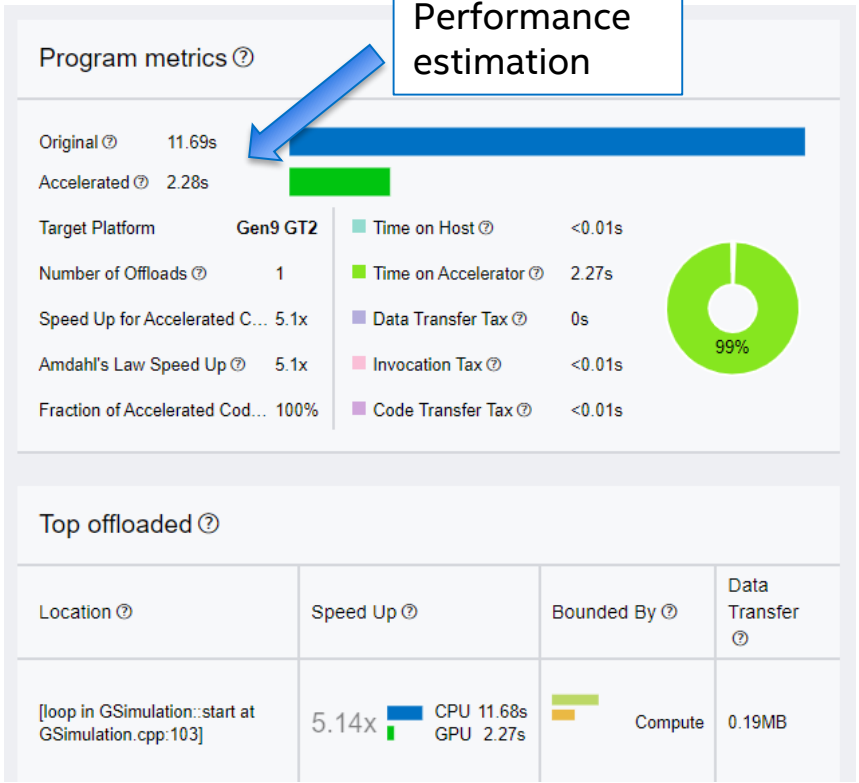
```
nbody-run:
  stage: test
  tags:
    - oneapi
  script:
    - ./nbody/base/nbody $NBODY_ARGS
    - SYCL_DEVICE_TYPE=GPU
./nbody/dpcpp/nbody $NBODY_ARGS
  dependencies:
    - nbody-base-build
    - nbody-dpcpp-build
```

ACTIVITY 6: COMPARING BASE AND DPC++ VERSIONS

Activity 6

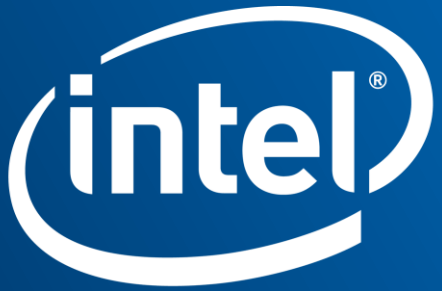
Actual runs

Performance estimation




```
$ ./nbody/base/nbody $NBODY_ARGS
================================
 Initialize Gravity Simulation
 nPart = 4000; nSteps = 1000; dt = 0.1
----------------------------------------
 s       dt      kenergy     time (s)
----------------------------------------
 50      5       66.863      0.67641
 100     10      369.19      0.58348
 150     15      1089.7      0.58435
 200     20      3005.8      0.58221
 250     25      15704       0.5829
 300     30      10062       0.57951
 350     35      6497.2      0.57881
 400     40      5674        0.58026
 450     45      5350.2      0.57886
 500     50      5019.5      0.58176
 550     55      5043.5      0.57916
 600     60      4815.3      0.57909
 650     65      5010.3      0.5809
 700     70      4783        0.57874
 750     75      4797.8      0.58029
 800     80      4987.9      0.57866
 850     85      4699.8      0.58215
 900     90      4534        0.57835
 950     95      4907.4      0.58682
 1000    100     4915.7      0.58003
# Total Time (s)      : 11.713
================================
```

```
$ ./nbody/dpcpp/nbody $NBODY_ARGS
================================
 Initialize Gravity Simulation
 nPart = 4000; nSteps = 1000; dt = 0.1
----------------------------------------
 s       dt      kenergy     time (s)
----------------------------------------
 50      5       66.863      0.345
 100     10      369.19      0.10216
 150     15      1089.7      0.10016
 200     20      3005.8      0.10037
 250     25      15704       0.10028
 300     30      10062       0.10016
 350     35      6497.2      0.10043
 400     40      5674        0.10045
 450     45      5350.2      0.10023
 500     50      5019.5      0.10045
 550     55      5043.5      0.099716
 600     60      4815.3      0.10027
 650     65      5010.3      0.10033
 700     70      4783        0.1006
 750     75      4797.8      0.1004
 800     80      4987.9      0.10037
 850     85      4699.8      0.10162
 900     90      4533.9      0.099533
 950     95      4907.5      0.1011
 1000    100     4915.7      0.10097
# Total Time (s)      : 2.2563
================================
```



Software

Notices & Disclaimers

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.

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