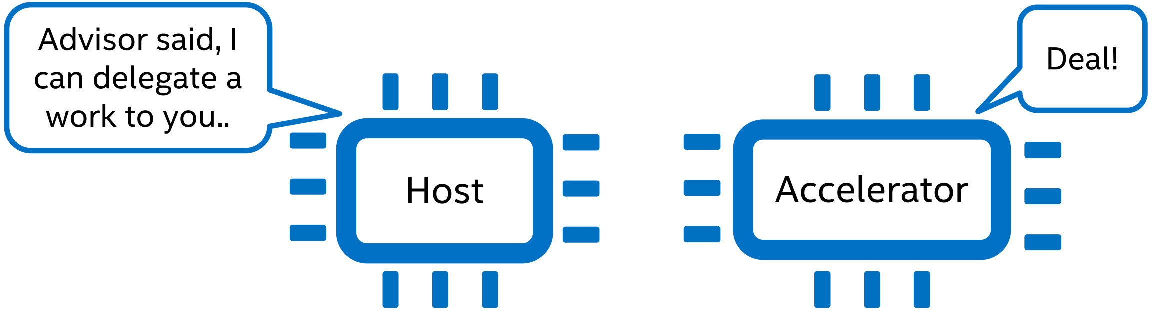
OFFLOAD ADVISOR

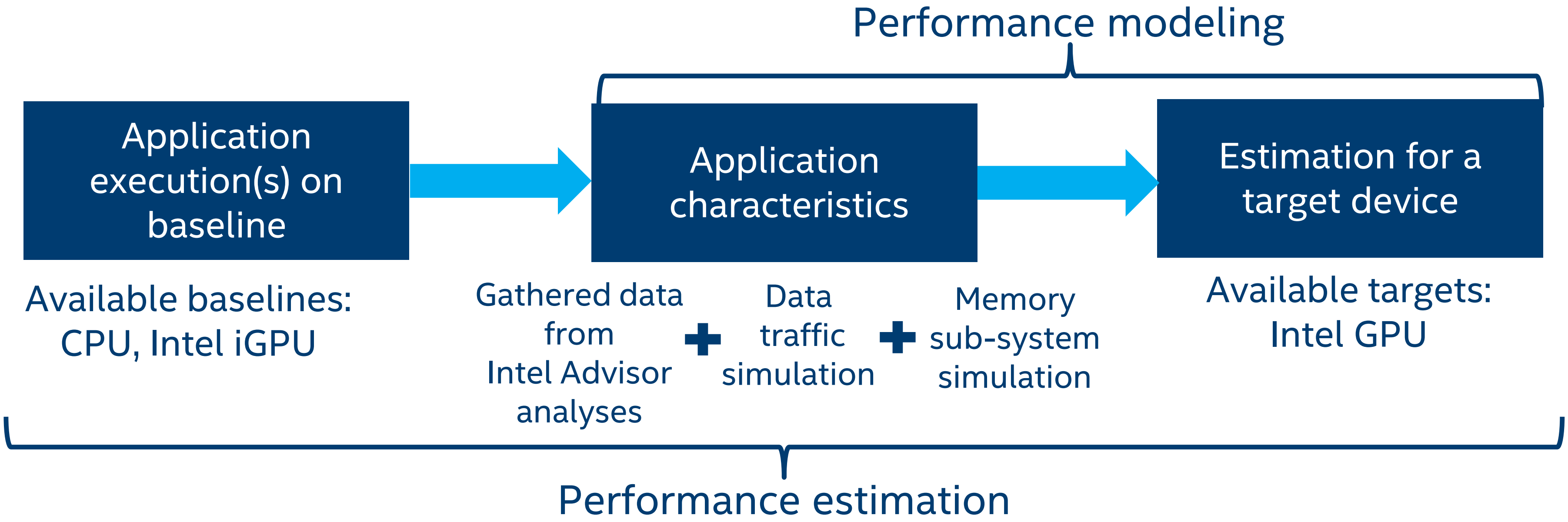
Purpose of Offload Advisor

- 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

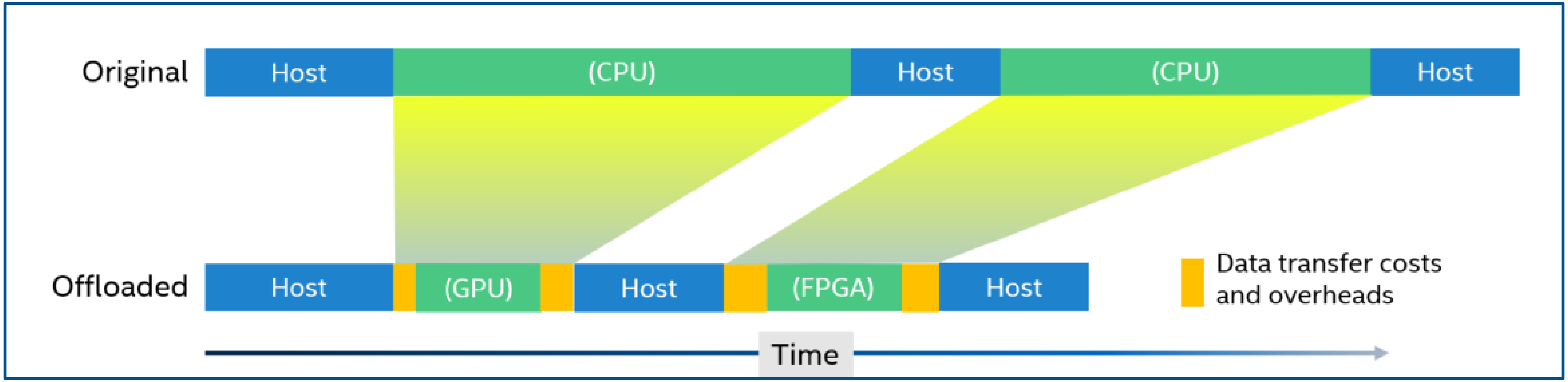


How it works

- 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)



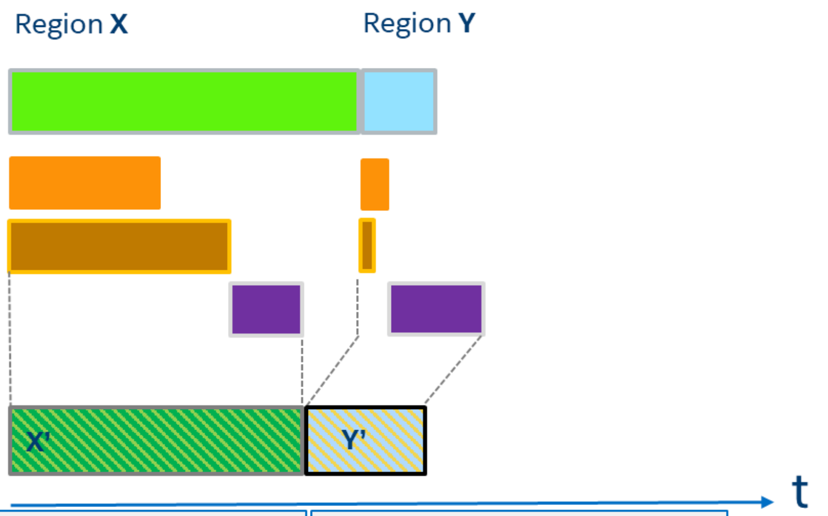
- Data transfer taxes: copy data between CPU and GPU
- Offload taxes: time to place task to GPU task dispatcher

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)



X – profitable to accelerate, $t(X) > t(X')$

Y - too much overhead, not accelerable, $t(Y) < t(Y')$

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

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)

$\boldsymbol{Objective\ function}$:

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

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)

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

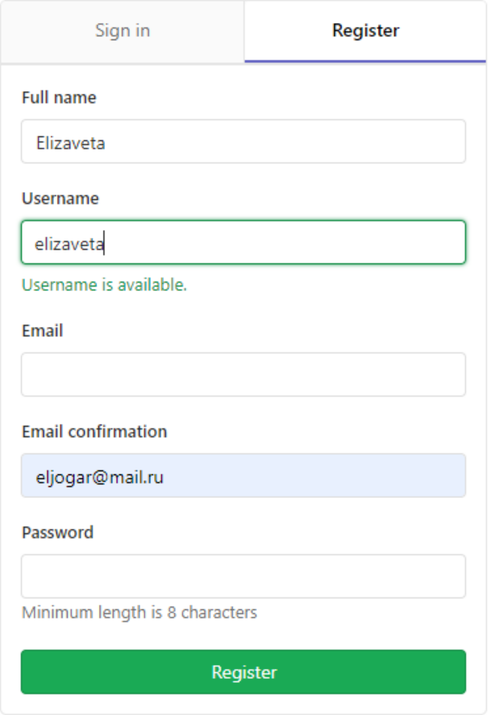
Activity 4: Rewriting code using DPC++

Activity 5: Building N-body DPC++ version

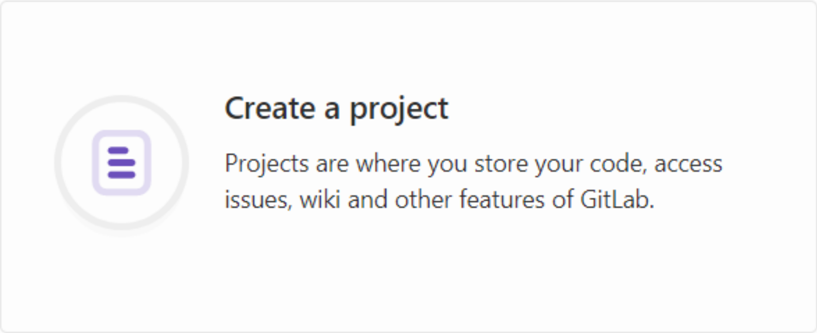
Activity 6: Comparing base and DPC++ versions

ACTIVITY 0: MAKING THE PROJECT

Activity 0: Make your Project

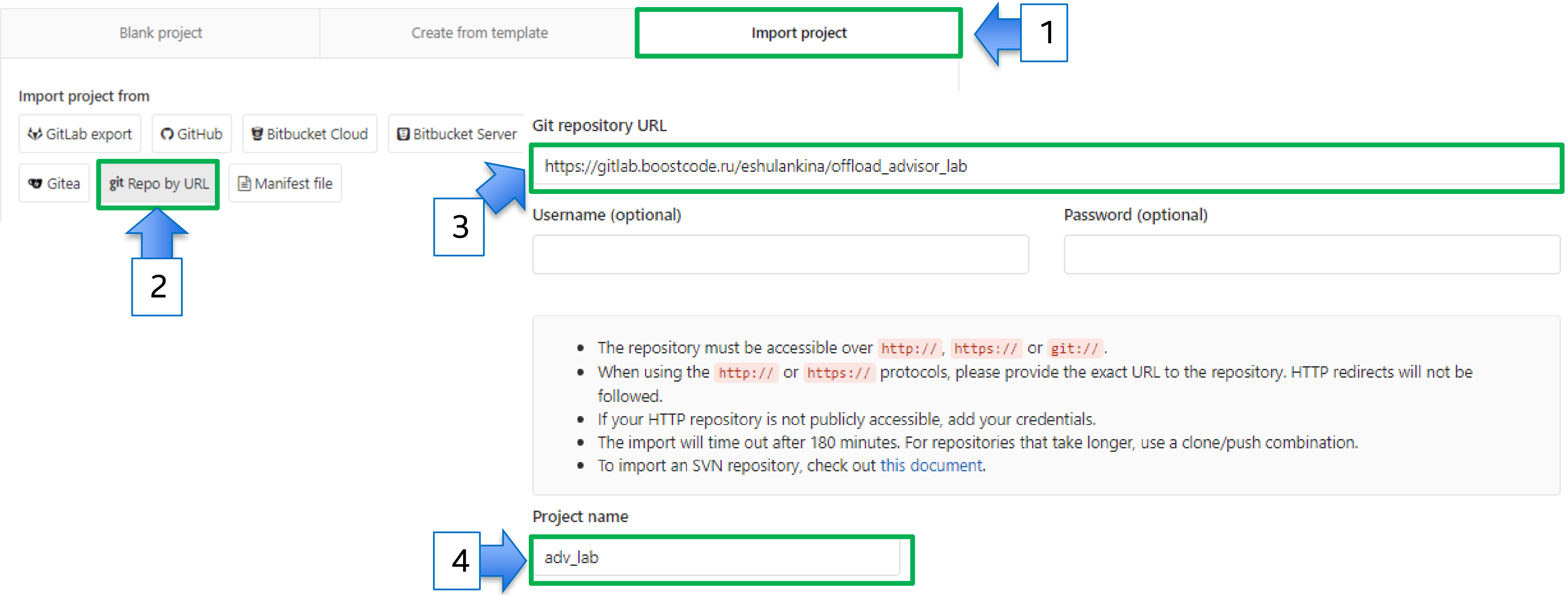


Navigate to https://gitlab.boostcode.ru/ , Register and Create a project



Activity 0: Import Project

Import from: https://gitlab.boostcode.ru/eshulankina/offload_advisor_lab



ACTIVITY 1: BUILDING N-BODY BASE VERSION

Activity 1: Build & Run

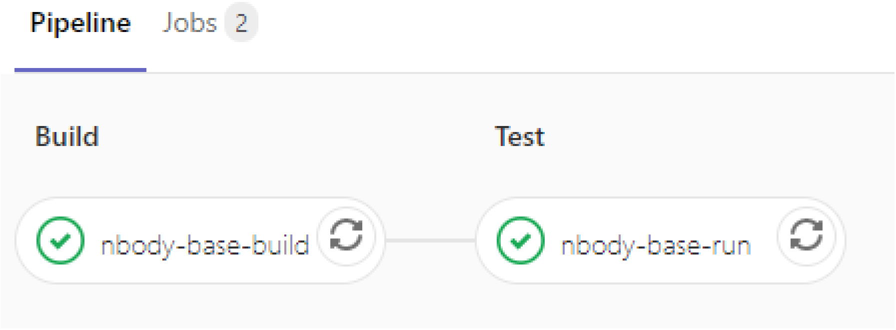
Create your own pipeline to build & run the N-Body base version:

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

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

```
nbody-base-run:
  stage: run
  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]`

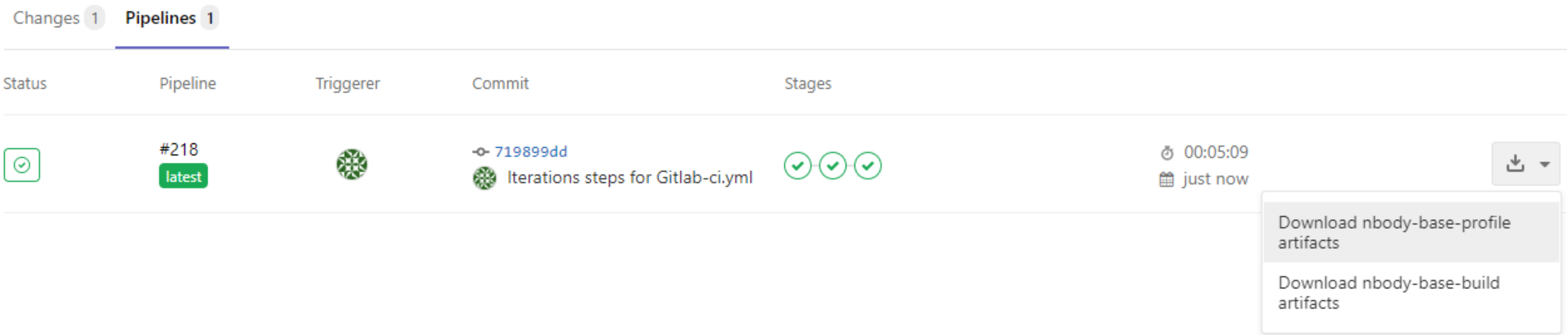
Activity 2

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

- Uncomment the following job:

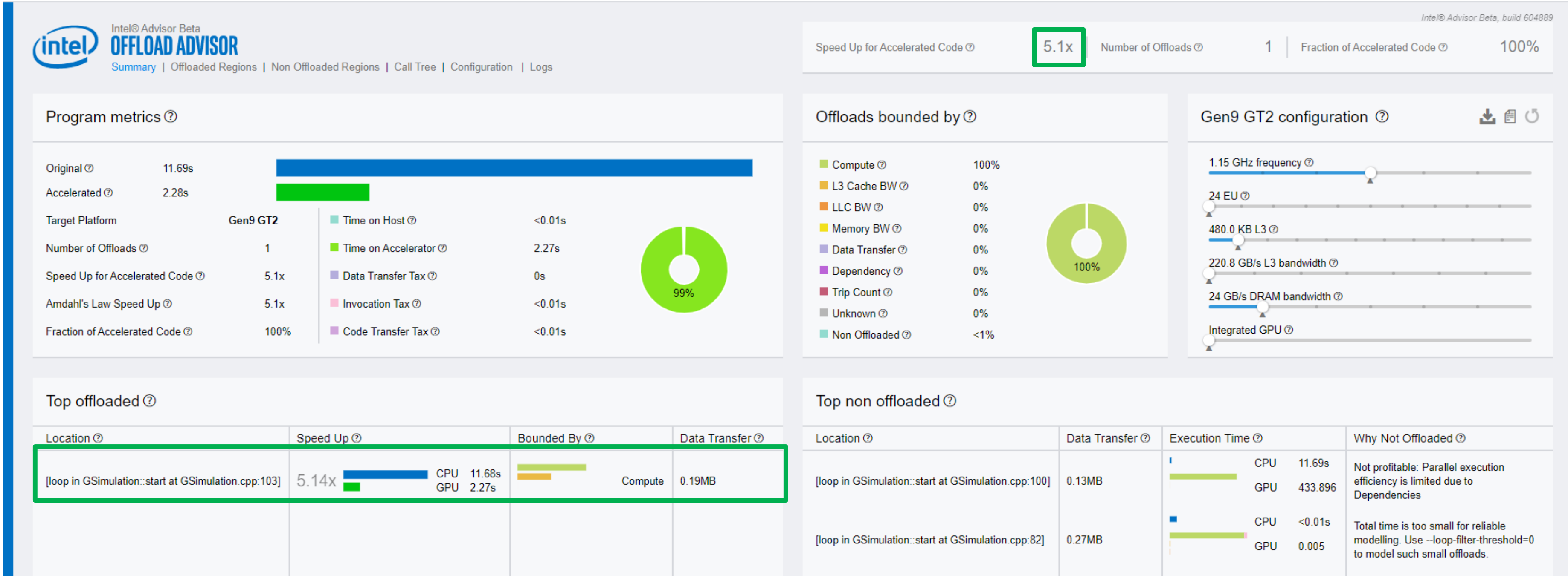
```
nbody-base-profile:
  stage: profile
  tags:
    - oneapi
  before_script:
    - . . .
  script:
    - mkdir adv_prj

    # 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
  . . .
```

Activity 2: Download artifacts



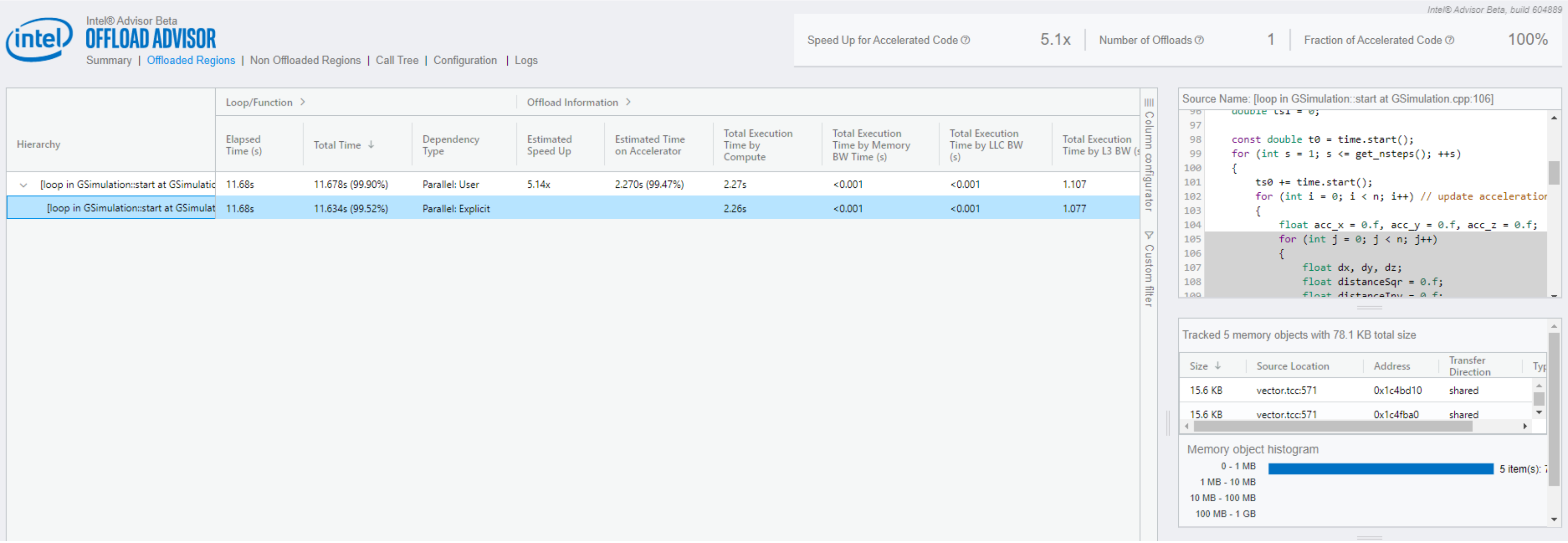
ACTIVITY 3: LOOKING AT THE ESTIMATION RESULTS

Activity 3



Intel® Advisor Beta, build 604889
Intel® Advisor Beta
OFFLOAD ADVISOR
Summary | Offloaded Regions | Non Offloaded Regions | Call Tree | Configuration | Logs
Speed Up for Accelerated Code 5.1x
Number of Offloads 1
Fraction of Accelerated Code 100%
Program metrics
Original 11.69s
Accelerated 2.28s
Target Platform Gen9 GT2
Number of Offloads 1
Speed Up for Accelerated Code 5.1x
Amdahl's Law Speed Up 5.1x
Fraction of Accelerated Code 100%
Time on Host <0.01s
Time on Accelerator 2.27s
Data Transfer Tax 0s
Invocation Tax <0.01s
Code Transfer Tax <0.01s
99%
Offloads bounded by
Compute 100%
L3 Cache BW 0%
LLC BW 0%
Memory BW 0%
Data Transfer 0%
Dependency 0%
Trip Count 0%
Unknown 0%
Non Offloaded <1%
100%
Gen9 GT2 configuration
1.15 GHz frequency
24 EU
480.0 KB L3
220.8 GB/s L3 bandwidth
24 GB/s DRAM bandwidth
Integrated GPU
Top offloaded
Location
Speed Up
Bounded By
Data Transfer
[loop in GSimulation::start at GSimulation.cpp:103]
5.14x
CPU 11.68s
GPU 2.27s
Compute
0.19MB
Top non offloaded
Location
Data Transfer
Execution Time
Why Not Offloaded
[loop in GSimulation::start at GSimulation.cpp:100]
0.13MB
CPU 11.69s
GPU 433.896
Not profitable: Parallel execution efficiency is limited due to Dependencies
[loop in GSimulation::start at GSimulation.cpp:82]
0.27MB
CPU <0.01s
GPU 0.005
Total time is too small for reliable modelling. Use --loop-filter-threshold=0 to model such small offloads.

Activity 3


Intel® Advisor Beta, build 604889
Intel® Advisor Beta
OFFLOAD ADVISOR
Summary | Offloaded Regions | Non Offloaded Regions | Call Tree | Configuration | Logs
Speed Up for Accelerated Code 5.1x
Number of Offloads 1
Fraction of Accelerated Code 100%
Hierarchy
Loop/Function
Offload Information
Elapsed Time (s)
Total Time
Dependency Type
Estimated Speed Up
Estimated Time on Accelerator
Total Execution Time by Compute
Total Execution Time by Memory BW Time (s)
Total Execution Time by LLC BW (s)
Total Execution Time by L3 BW
[loop in GSimulation::start at GSimulatic 11.68s 11.678s (99.90%) Parallel: User 5.14x 2.270s (99.47%) 2.27s <0.001 <0.001 1.107
[loop in GSimulation::start at GSimulat 11.68s 11.634s (99.52%) Parallel: Explicit 2.26s <0.001 <0.001 1.077
Column configurator
Custom filter
Source Name: [loop in GSimulation::start at GSimulation.cpp:106]
const double t0 = time.start();
for (int s = 1; s <= get_nsteps(); ++s)
{
ts0 += time.start();
for (int i = 0; i < n; i++) // update acceleratior
{
float acc_x = 0.f, acc_y = 0.f, acc_z = 0.f;
for (int j = 0; j < n; j++)
{
float dx, dy, dz;
float distanceSqr = 0.f;
Tracked 5 memory objects with 78.1 KB total size
Size
Source Location
Address
Transfer Direction
15.6 KB vector.tcc:571 0x1c4bd10 shared
15.6 KB vector.tcc:571 0x1c4fba0 shared
Memory object histogram
0 - 1 MB
1 MB - 10 MB
10 MB - 100 MB
100 MB - 1 GB
5 item(s):

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:

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

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

ACTIVITY 6: COMPARING BASE AND DPC++ VERSIONS

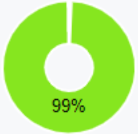
Activity 6

Performance estimation

Program metrics ⓘ

Original ⓘ	11.69s	
Accelerated ⓘ	2.28s	

Target Platform	Gen9 GT2	Time on Host ⓘ	<0.01s
Number of Offloads ⓘ	1	Time on Accelerator ⓘ	2.27s
Speed Up for Accelerated C...	5.1x	Data Transfer Tax ⓘ	0s
Amdahl's Law Speed Up ⓘ	5.1x	Invocation Tax ⓘ	<0.01s
Fraction of Accelerated Cod...	100%	Code Transfer Tax ⓘ	<0.01s



Top offloaded ⓘ

Location ⓘ	Speed Up ⓘ	Bounded By ⓘ	Data Transfer ⓘ
[loop in GSimulation::start at GSimulation.cpp:103]	5.14x CPU 11.68s GPU 2.27s	Compute	0.19MB

Actual runs

```
$ ./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.

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