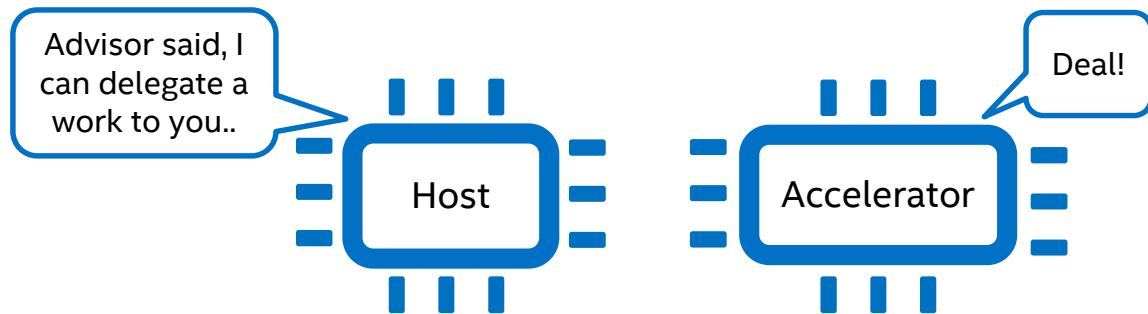




# OFFLOAD ADVISOR

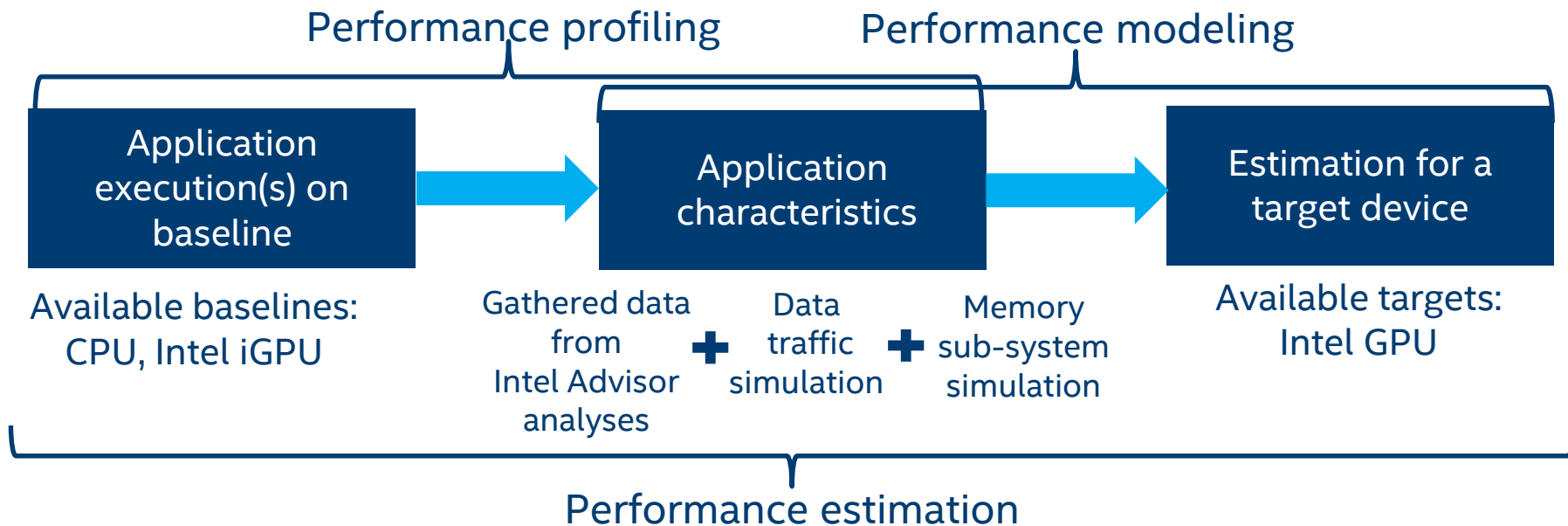
# Purpose of Offload Advisor

- 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

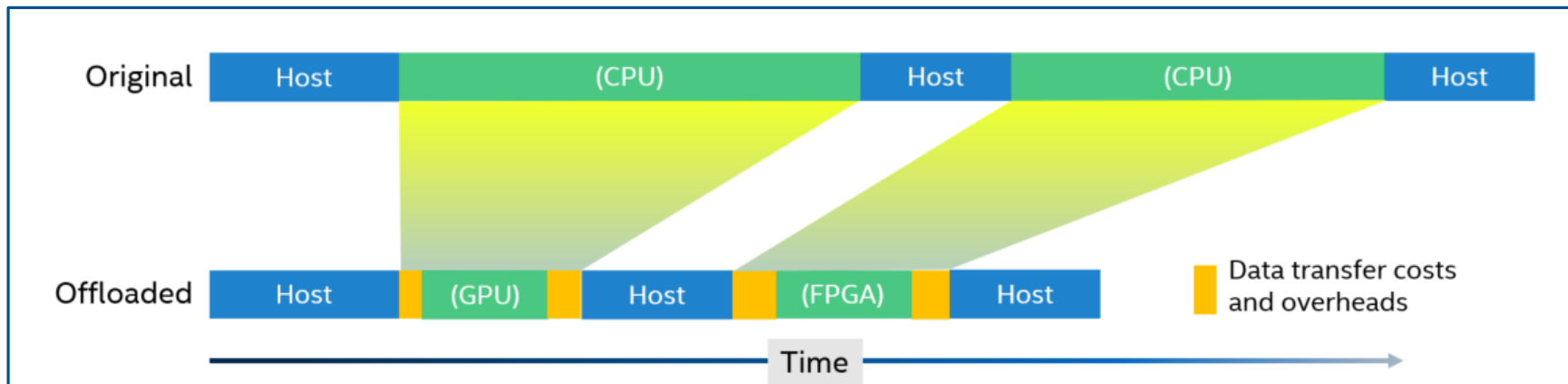


# How it works

- 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)

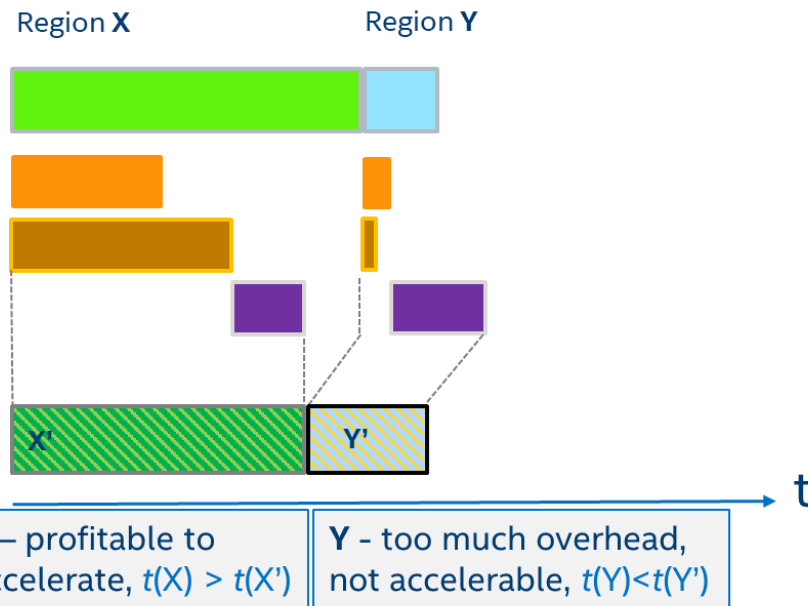


- Data transfer taxes: copy data between CPU and GPU
- Offload taxes: time to place task to GPU task dispatcher

# 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)

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

# 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 :**

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

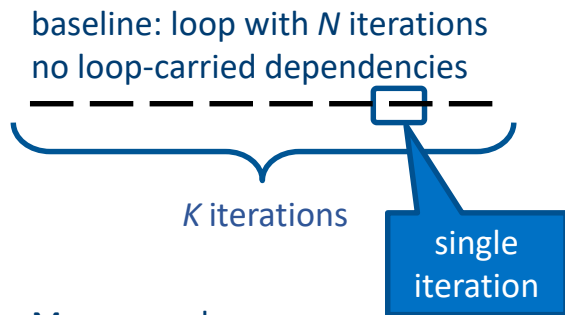
Reject loop nests for which  $T(x86) / T_{all}(x86+\"X\") < 1.0$

$$T_i = \max \left\{ \begin{array}{l} T_i^{Comp\_only} \\ T_i^{M_k-only} (M_i^k) = \frac{M_i^k}{BW_k} \end{array} \right.$$

This is effective “balance” (throughput) model

*Under algorithmic constraints (Dependencies and TripCount/Granularity)*

# How it works: GPU Execution Time Model



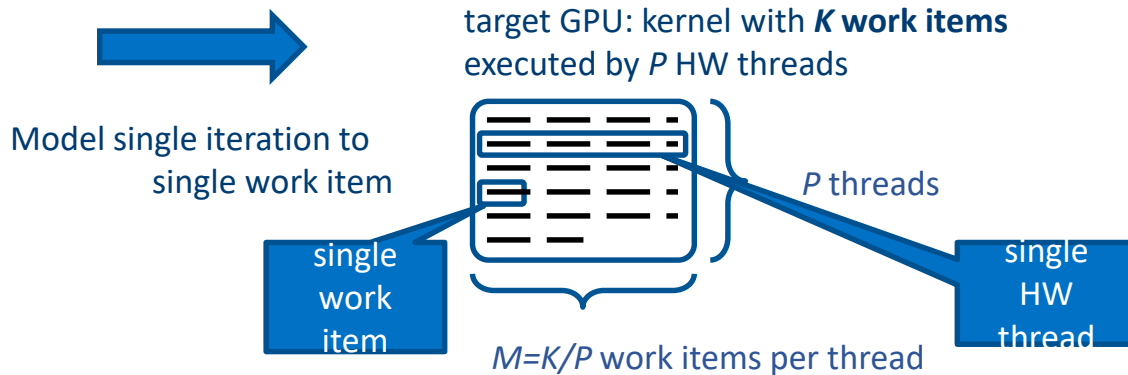
More complex cases:

- 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



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

$$M = \text{ceil}(K/P)$$

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

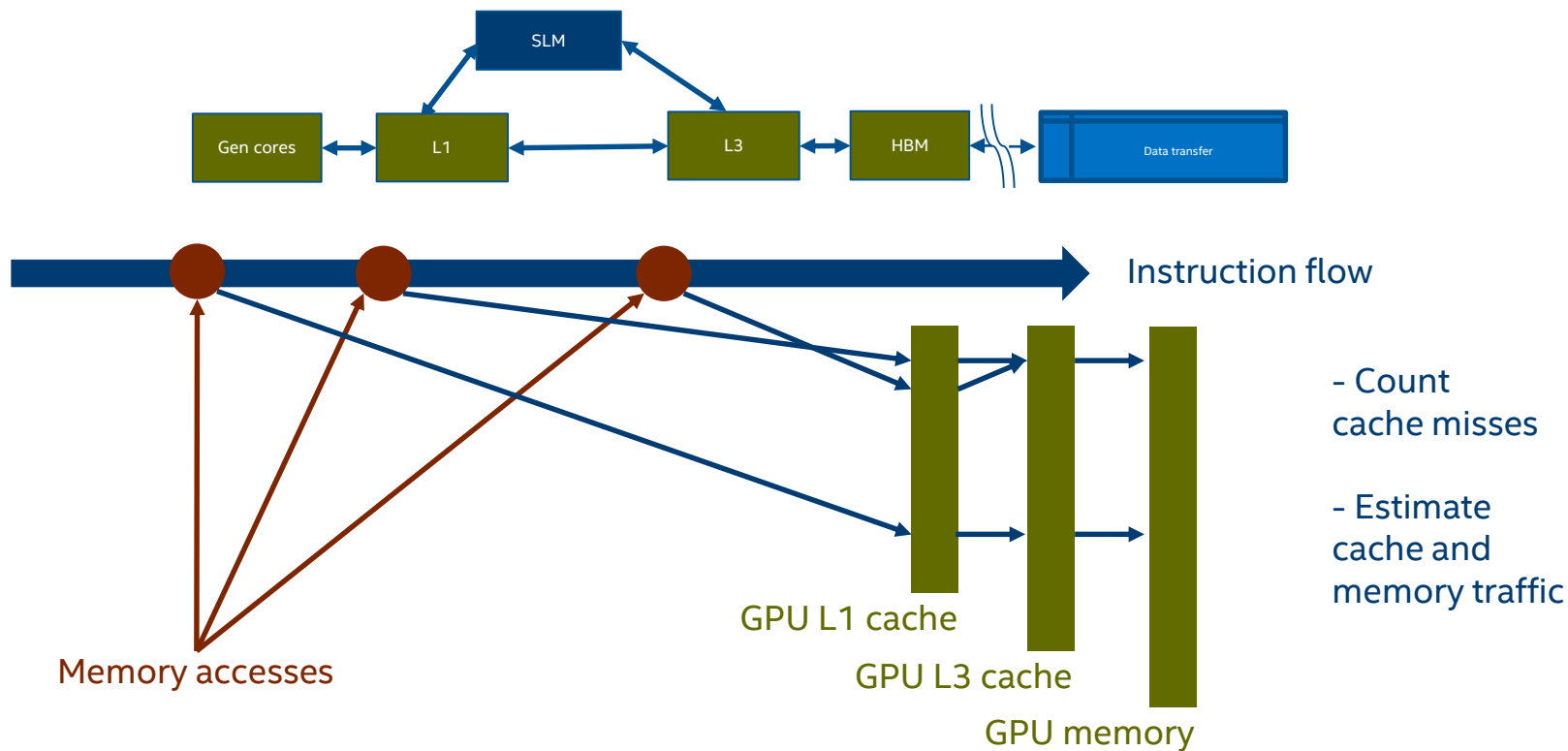
cost of single loop iteration (in GPU cycles)

$C_i$  - 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





# 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

Sign in Register

Full name

Username

Username is available.

Email

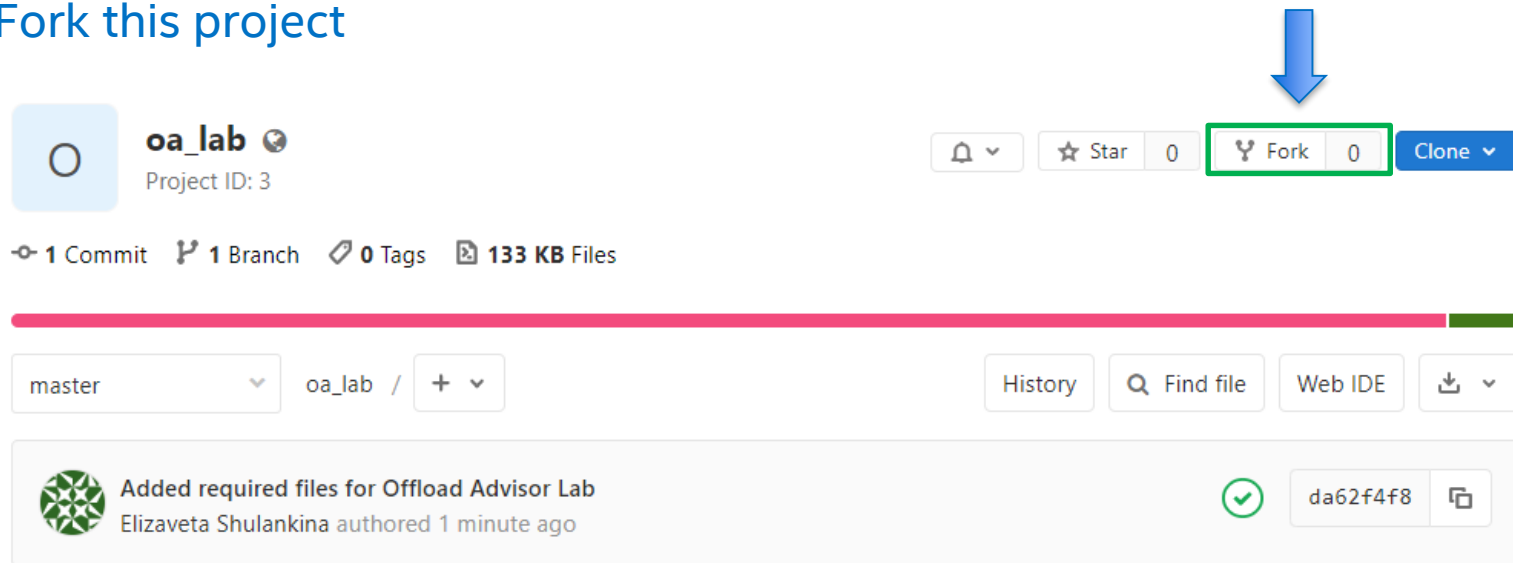
Email confirmation

Password

Minimum length is 8 characters

# Activity 0: Fork a Project

1. Navigate to [https://oneapi.team/dsivkov/offload\\_advisor](https://oneapi.team/dsivkov/offload_advisor)
2. Fork this project



The screenshot shows the GitHub interface for the repository 'oa\_lab'. The repository name is 'oa\_lab' with a Project ID of 3. It has 1 commit, 1 branch, 0 tags, and 133 KB of files. The 'Fork' button is highlighted with a green border and a blue arrow pointing down to it. Below the repository name, there are buttons for 'History', 'Find file', 'Web IDE', and a download icon. A commit message is visible: 'Added required files for Offload Advisor Lab' by Elizaveta Shulankina, authored 1 minute ago, with a commit hash of da62f4f8.

# 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](#)

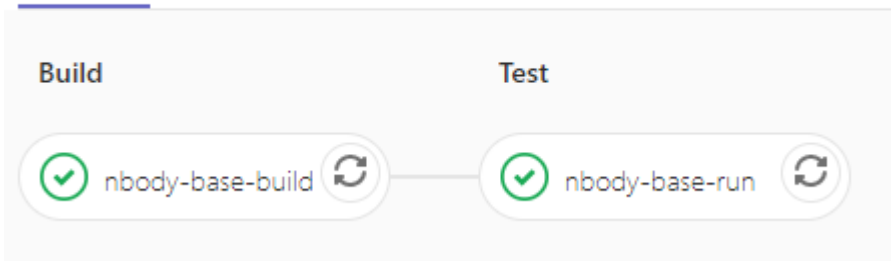
```
nbody-base-build:
  stage: build
  tags:
    - oneapi
  script:
    - make -C ./nbody/base
  artifacts:
    paths:
      - ./nbody/base/nbody
```

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

Pipeline Jobs 2



```
$ ./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 <b>run_oa.py</b> script	Most simple and least flexible. <b>Does not support MPI applications.</b>
Combination of <b>collect.py</b> and <b>analyze.py</b> scripts	Somewhat simple and flexible. <b>Does not support MPI applications.</b>
Combination of Advisor Command Line <b>advixe-cl</b> and <b>analyze.py</b> script	Least simple and most flexible. <b>Applicable to MPI applications.</b>

# 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
- advise-python $APM/collect.py --config=gen9 ./adv_prj -c basic -- ./nbody/base/nbody $NBODY_ARGS
- advise-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:

<code>-c basic</code>	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:

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



# Activity 2: Download artifacts

Changes 1 Pipelines 1

Status	Pipeline	Triggerer	Commit	Stages		
	#218 latest		<a href="#">719899dd</a> Iterations steps for Gitlab-ci.yml	  	🕒 00:05:09 📅 just now	

- Download nbody-base-profile artifacts
- Download nbody-base-build artifacts

# **ACTIVITY 3: LOOKING AT THE ESTIMATION RESULTS**

# Activity 3



Intel® Advisor Beta

**OFFLOAD ADVISOR**

[Summary](#) | [Offloaded Regions](#) | [Non Offloaded Regions](#) | [Call Tree](#) | [Configuration](#) | [Logs](#)

Intel® Advisor Beta, build 604989

Speed Up for Accelerated Code [?](#)

5.1x

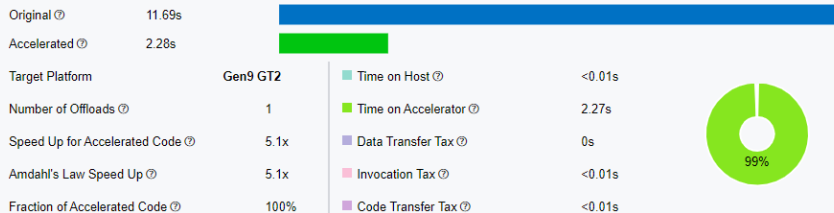
Number of Offloads [?](#)

1

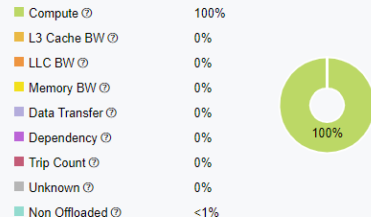
Fraction of Accelerated Code [?](#)

100%

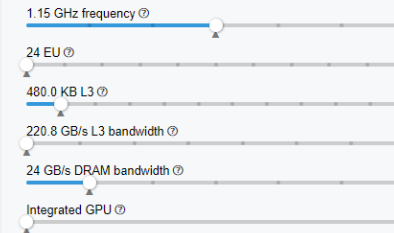
## Program metrics [?](#)



## Offloads bounded by [?](#)



## Gen9 GT2 configuration [?](#)




## Top offloaded [?](#)

Location <a href="#">?</a>	Speed Up <a href="#">?</a>	Bounded By <a href="#">?</a>	Data Transfer <a href="#">?</a>
[loop in GSimulation::start at GSimulation.cpp:103]	5.14x	Compute	0.19MB

## Top non offloaded [?](#)

Location <a href="#">?</a>	Data Transfer <a href="#">?</a>	Execution Time <a href="#">?</a>	Why Not Offloaded <a href="#">?</a>
[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  
**OFFLOAD ADVISOR**

Summary | [Offloaded Regions](#) | [Non Offloaded Regions](#) | [Call Tree](#) | [Configuration](#) | [Logs](#)

Intel® Advisor Beta, build 604889

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 (s)
▼ [loop in GSimulation::start at GSimulation.cpp:106]	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 GSimulation.cpp:106]	11.68s	11.634s (99.52%)	Parallel: Explicit			2.26s	<0.001	<0.001	1.077

**Source Name:** [loop in GSimulation::start at GSimulation.cpp:106]

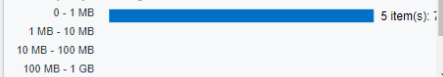
```

97  double tsz = 0;
98
99  const double t0 = time.start();
100  for (int s = 1; s <= get_nsteps(); ++s)
101  {
102      ts0 += time.start();
103      for (int i = 0; i < n; i++) // update acceleration
104      {
105          float acc_x = 0.f, acc_y = 0.f, acc_z = 0.f;
106          for (int j = 0; j < n; j++)
107          {
108              float dx, dy, dz;
109              float distanceSqr = 0.f;
110              float distanceTrj = 0.f;
          
```

Tracked 5 memory objects with 78.1 KB total size

Size ↓	Source Location	Address	Transfer Direction	Type
15.6 KB	vector.tcc:571	0x1c4bd10	shared	
15.6 KB	vector.tcc:571	0x1c4ba0	shared	

Memory object histogram



# ACTIVITY 4: REWRITING CODE USING DPC++

# Activity 4

```
1 const double t0 = time.start();
2 for (int s = 1; s <= get_nsteps(); ++s)
3 {
4     ts0 += time.start();
5     for (int i = 0; i < n; i++) // update acceleration
6     {
7         float acc_x = 0.f, acc_y = 0.f, acc_z = 0.f;
8         for (int j = 0; j < n; j++)
9         {
10            . . .
11
12            dx = particles.pos_x[j] - particles.pos_x[i];
13            dy = particles.pos_y[j] - particles.pos_y[i];
14            dz = particles.pos_z[j] - particles.pos_z[i];
15
16            distanceSqr = dx * dx + dy * dy + dz * dz + softeningSquared;
17            distanceInv = 1.0f / sqrtf(distanceSqr);
18
19            acc_x += dx * G * particles.mass[j] * distanceInv * distanceInv * distanceInv;
20            acc_y += dy * G * particles.mass[j] * distanceInv * distanceInv * distanceInv;
21            acc_z += dz * G * particles.mass[j] * distanceInv * distanceInv * distanceInv;
22        }
23        particles.acc_x[i] = acc_x;
24        particles.acc_y[i] = acc_y;
25        particles.acc_z[i] = acc_z;
26    }
27    energy = 0.f;
```

```
1  sycl::default_selector selector;
2  sycl::queue queue(selector, exceptionHandler);
3
4  sycl::buffer<float, 1> posX(particles.pos_x.data(), sycl::range<1>(n));
5  sycl::buffer<float, 1> posY(particles.pos_y.data(), sycl::range<1>(n));
6  sycl::buffer<float, 1> posZ(particles.pos_z.data(), sycl::range<1>(n));
7  sycl::buffer<float, 1> velX(particles.vel_x.data(), sycl::range<1>(n));
8  . . .
9  sycl::buffer<float, 1> mass(particles.mass.data(), sycl::range<1>(n));
10 sycl::buffer<float, 1> energy(_energy, sycl::range<1>(n));
11
12 const double t0 = time.start();
13 for (int s = 1; s <= get_nsteps(); ++s)
14 {
15     ts0 += time.start();
16     queue.submit([&](sycl::handler &cgh) {
17         auto posXBuff = posX.get_access<sycl::access::mode::read>(cgh);
18         auto posYBuff = posY.get_access<sycl::access::mode::read>(cgh);
19         auto posZBuff = posZ.get_access<sycl::access::mode::read>(cgh);
20         auto massBuff = mass.get_access<sycl::access::mode::read>(cgh);
21         auto accXBuff = accX.get_access<sycl::access::mode::write>(cgh);
22         auto accYBuff = accY.get_access<sycl::access::mode::write>(cgh);
23         auto accZBuff = accZ.get_access<sycl::access::mode::write>(cgh);
24         cgh.parallel_for<class kernel1>(sycl::range<1>(n), [=](sycl::id<1> index) {
25             int i = index.get(0);
26             float acc_x = 0.f, acc_y = 0.f, acc_z = 0.f;
27             for (int j = 0; j < n; j++)
28             {
29                 . . .
30
31                 dx = posXBuff[j] - posXBuff[i];
32                 dy = posYBuff[j] - posYBuff[i];
33                 dz = posZBuff[j] - posZBuff[i];
34
35                 distanceSqr = dx * dx + dy * dy + dz * dz + softeningSquared;
36                 distanceInv = 1.0f / sycl::sqrt(distanceSqr);
37
38                 acc_x += dx * G * massBuff[j] * distanceInv * distanceInv * distanceInv;
39                 acc_y += dy * G * massBuff[j] * distanceInv * distanceInv * distanceInv;
40                 acc_z += dz * G * massBuff[j] * distanceInv * distanceInv * distanceInv;
41             }
42             accXBuff[i] = acc_x;
43             accYBuff[i] = acc_y;
44             accZBuff[i] = acc_z;
45         });
46     }).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

## Program metrics ?

Original ? 11.69s

Accelerated ? 2.28s

Target Platform Gen9 GT2

Number of Offloads ? 1

Speed Up for Accelerated C... 5.1x

Amdahl's Law Speed Up ? 5.1x

Fraction of Accelerated Cod... 100%

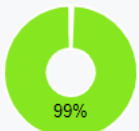
Time on Host ? <0.01s

Time on Accelerator ? 2.27s

Data Transfer Tax ? 0s

Invocation Tax ? <0.01s

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

```
./nbody/base/nbody $NBODY_ARGS
```

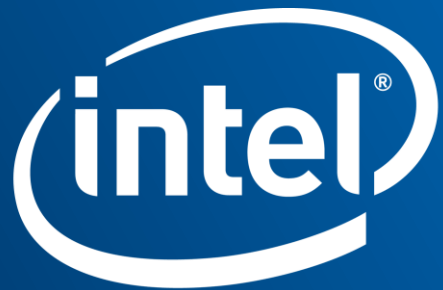
```
Initialize Gravity Simulation
nPart = 4000; nSteps = 1000; dt = 0.1
```

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](http://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](http://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 ©, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Core, VTune, OpenVINO, and the Intel logo are trademarks of Intel Corporation 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