Managing Accelerated Application Memory with CUDA C/C++ Unified Memory

CUDA

The *CUDA Best Practices Guide*, a highly recommended followup to this and other CUDA fundamentals labs, recommends a design cycle called **APOD**: **A**ssess, **P**arallelize, **O**ptimize, **D**eploy. In short, APOD prescribes an iterative design process, where developers can apply incremental improvements to their accelerated application's performance, and ship their code. As developers become more competent CUDA programmers, more advanced optimization techniques can be applied to their accelerated code bases.

This lab will support such a style of iterative development. You will be using the Nsight Systems command line tool **nsys** to qualitatively measure your application's performance, and to identify opportunities for optimization, after which you will apply incremental improvements before learning new techniques and repeating the cycle. As a point of focus, many of the techniques you will be learning and applying in this lab will deal with the specifics of how CUDA's **Unified Memory** works. Understanding Unified Memory behavior is a fundamental skill for CUDA developers, and serves as a prerequisite to many more advanced memory management techniques.

Prerequisites

To get the most out of this lab you should already be able to:

- Write, compile, and run C/C++ programs that both call CPU functions and launch GPU kernels.
- Control parallel thread hierarchy using execution configuration.
- Refactor serial loops to execute their iterations in parallel on a GPU.
- Allocate and free Unified Memory.

Objectives

By the time you complete this lab, you will be able to:

- Use the Nsight Systems command line tool (**nsys**) to profile accelerated application performance.
- Leverage an understanding of **Streaming Multiprocessors** to optimize execution configurations.

- Understand the behavior of **Unified Memory** with regard to page faulting and data migrations.
- Use **asynchronous memory prefetching** to reduce page faults and data migrations for increased performance.
- Employ an iterative development cycle to rapidly accelerate and deploy applications.

Iterative Optimizations with the NVIDIA Command Line Profiler

The only way to be assured that attempts at optimizing accelerated code bases are actually successful is to profile the application for quantitative information about the application's performance. `nsys` is the Nsight Systems command line tool. It ships with the CUDA toolkit, and is a powerful tool for profiling accelerated applications.

`nsys` is easy to use. Its most basic usage is to simply pass it the path to an executable compiled with `nvcc`. `nsys` will proceed to execute the application, after which it will print a summary output of the application's GPU activities, CUDA API calls, as well as information about **Unified Memory** activity, a topic which will be covered extensively later in this lab.

When accelerating applications, or optimizing already-accelerated applications, take a scientific and iterative approach. Profile your application after making changes, take note, and record the implications of any refactoring on performance. Make these observations early and often: frequently, enough performance boost can be gained with little effort such that you can ship your accelerated application. Additionally, frequent profiling will teach you how specific changes to your CUDA code bases impact its actual performance: knowledge that is hard to acquire when only profiling after many kinds of changes in your code bases.

Exercise: Profile an Application with nsys

01-vector-add.cu (<------ you can click on this and any of the source file links in this lab to open them for editing) is a naively accelerated vector addition program. Use the two code execution cells below (`CTRL` + `ENTER`). The first code execution cell will compile (and run) the vector addition program. The second code execution cell will profile the executable that was just compiled using `nsys profile`.

`nsys profile` will generate a report file which can be used in a variety of manners, including for use in visual profiling with Nsight Systems, which we will look at in more detail in the following section.

Here we use the `--stats=true` flag to indicate we would like summary statistics printed. In this section this summary will be the focus of our attention. There is quite a lot of information printed:

- Operating System Runtime Summary (`osrt_sum`)
- **CUDA API Summary (`cuda_api_sum`)**
- **CUDA Kernel Summary (`cuda_gpu_kern_sum`)**
- **CUDA Memory Time Operation Summary (`cuda_gpu_mem_time_sum`)**
- **CUDA Memory Size Operation Summary (`cuda_gpu_mem_size_sum`)**

In this section you will primarily be using the 4 summaries in **bold** above. In the next section, you will be using the generated report files to give to the Nsight Systems GUI for visual profiling.

After profiling the application, answer the following questions using information displayed in the `cuda_gpu_kern_sum` section of the profiling output:

- What was the name of the only CUDA kernel called in this application?
- How many times did this kernel run?
- How long did it take this kernel to run? Record this time somewhere: you will be optimizing this application and will want to know how much faster you can make it.

In [3]:
```
!nvcc -o single-thread-vector-add 01-vector-add/01-vector-add.cu -run
```

```
Success! All values calculated correctly.
```

In [4]:
```
!nsys profile --stats=true ./single-thread-vector-add
```

```
Success! All values calculated correctly.
Generating '/tmp/nsys-report-14fa.qdstrm'
[1/8] [========================100%] report2.nsys-rep
[2/8] [========================100%] report2.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report2.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     90.3       6053933434        317  19097581.8  10071878.0      1840  100154357
27332108.1  poll
      8.9        594680679        283   2101345.2   2064414.0       120   20429370
1424518.7  sem_timedwait
      0.5         31671327        499     63469.6     11150.0       380    7939262
377485.5  ioctl
      0.3         19865890         24    827745.4      4615.0       850    7192390
2227989.6  mmap
      0.0           881923         27     32663.8      4410.0      2990     537309
101683.8  mmap64
      0.0           478318         44     10870.9     10825.0      3600      30821
4430.2  open64
      0.0           165352          4     41338.0     38525.5     31220      57081
12035.9  pthread_create
      0.0           142572         11     12961.1     13700.0       840      18030
4639.4  write
      0.0           134802         29      4648.3      3221.0      1480      20720
4117.7  fopen
      0.0            60743         11      5522.1      3510.0      1701      18081
5120.5  munmap
      0.0            50851         26      1955.8        70.0        60      49081
9611.7  fgets
      0.0            35851         52       689.4       520.0       160       6510
872.7  fcntl
      0.0            33091          6      5515.2      5515.0      2500       7960
2171.1  open
      0.0            25330         22      1151.4       970.0       550       3770
672.1  fclose
      0.0            21741         14      1552.9      1205.5       780       4050
998.4  read
      0.0            16710          2      8355.0      8355.0      5490      11220
4051.7  socket
      0.0            12200          1     12200.0     12200.0     12200      12200
0.0  connect
      0.0             8081          5      1616.2      1351.0        70       3300
1473.5  fread
      0.0             6230          1      6230.0      6230.0      6230       6230
0.0  pipe2
      0.0             5290         64        82.7        50.0        40        190
46.6  pthread_mutex_trylock
      0.0             2540          1      2540.0      2540.0      2540       2540
0.0  bind
      0.0             1210          1      1210.0      1210.0      1210       1210
0.0  listen
      0.0              280          1       280.0       280.0       280        280
0.0  pthread_cond_broadcast

[5/8] Executing 'cuda_api_sum' stats report
```

```
 Time (%)  Total Time (ns)  Num Calls    Avg (ns)      Med (ns)     Min (ns)    Max
(ns)   StdDev (ns)          Name
 --------  ---------------  ---------  ------------  ------------  ----------  ------
----  -----------  --------------------
     95.1       2471506658          1  2471506658.0  2471506658.0  2471506658  247150
6658          0.0  cudaDeviceSynchronize
      4.2        108272853          3    36090951.0       30510.0       16091   10822
6252   62471003.6  cudaMallocManaged
      0.8         19932312          3     6644104.0     6577719.0     6118312     723
6281     561933.2  cudaFree
      0.0            46621          1       46621.0       46621.0       46621       4
6621          0.0  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances    Avg (ns)      Med (ns)     Min (ns)    Max
(ns)   StdDev (ns)                     Name
 --------  ---------------  ---------  ------------  ------------  ----------  ------
----  -----------  -----------------------------------------
    100.0       2471496420          1  2471496420.0  2471496420.0  2471496420  247149
6420          0.0  addVectorsInto(float *, float *, float *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
     75.5         34138263   2304   14817.0    4351.5      1982     80224      22490.
4  [CUDA Unified Memory memcpy HtoD]
     24.5         11062385    768   14404.1    3759.5      1279     80544      22784.
2  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    402.653   2304     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy HtoD]
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report2.nsys-rep
    /dli/task/report2.sqlite
```

Worth mentioning is that by default, `nsys profile` will not overwrite an existing report file. This is done to prevent accidental loss of work when profiling. If for any reason, you would rather overwrite an existing report file, say during rapid iterations, you can provide the `-f` flag to `nsys profile` to allow overwriting an existing report file.

Exercise: Optimize and Profile

Take a minute or two to make a simple optimization to 01-vector-add.cu by updating its execution configuration so that it runs on many threads in a single thread block. Recompile and then profile with `nsys profile --stats=true` using the code execution cells below. Use the profiling output to check the runtime of the kernel. What was the speed up from this optimization? Be sure to record your results somewhere.

In [1]:
```
!nvcc -o multi-thread-vector-add 01-vector-add/01-vector-add.cu -run
```

```
Success! All values calculated correctly.
```

In [2]:
```
!nsys profile --stats=true ./multi-thread-vector-add
```

```
Success! All values calculated correctly.
Generating '/tmp/nsys-report-c4e6.qdstrm'
[1/8] [========================100%] report1.nsys-rep
[2/8] [========================100%] report1.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report1.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     90.4       6154343273        318  19353280.7  10074122.0      2480  100175395
27648662.0  poll
      8.7        594302075        283   2100007.3   2065185.0       120   20480740
1370120.5  sem_timedwait
      0.5         35976679        499     72097.6     12410.0       510    9199323
436415.2  ioctl
      0.3         19171110         24    798796.3      8840.0       820    7195710
2148526.1  mmap
      0.0          1101110         27     40781.9      4060.0      3210     705562
133858.6  mmap64
      0.0           496951         44     11294.3     10960.0      4460      33071
4281.1  open64
      0.0           181581         29      6261.4      3570.0      1500      46850
8528.5  fopen
      0.0           138540          4     34635.0     34565.0     24430      44980
10685.7  pthread_create
      0.0           130232         11     11839.3     11490.0       990      17751
5433.0  write
      0.0            99280         11      9025.5      4500.0      1590      35250
11560.4  munmap
      0.0            66782          6     11130.3      8325.0      3281      30851
9900.2  open
      0.0            57311         26      2204.3        80.0        70      55171
10803.1  fgets
      0.0            35690         52       686.3       510.0       210       5570
759.6  fcntl
      0.0            29470         22      1339.5      1195.0       710       3170
602.6  fclose
      0.0            23011         14      1643.6      1520.0       440       4060
1088.7  read
      0.0            16140          2      8070.0      8070.0      4200      11940
5473.0  socket
      0.0            12771          1     12771.0     12771.0     12771      12771
0.0  connect
      0.0            12431          5      2486.2      1541.0        70       6490
2766.4  fread
      0.0             6680          1      6680.0      6680.0      6680       6680
0.0  pipe2
      0.0             6090         64        95.2        85.0        40        360
60.5  pthread_mutex_trylock
      0.0             2330          1      2330.0      2330.0      2330       2330
0.0  bind
      0.0             1290          1      1290.0      1290.0      1290       1290
0.0  listen
      0.0              380          1       380.0       380.0       380        380
0.0  pthread_cond_broadcast

[5/8] Executing 'cuda_api_sum' stats report
```

```
 Time (%)  Total Time (ns)  Num Calls    Avg (ns)      Med (ns)     Min (ns)    Max
(ns)   StdDev (ns)          Name
 --------  ---------------  ---------  ------------  ------------  ----------  ------
----  -----------  ---------------------
     94.5       2468317502          1  2468317502.0  2468317502.0  2468317502  246831
7502          0.0  cudaDeviceSynchronize
      4.8        125277243          3    41759081.0       75602.0       18290  12518
3351   72247542.8  cudaMallocManaged
      0.7         19214830          3     6404943.3     6102102.0     5862067     725
0661     742181.2  cudaFree
      0.0            41700          1       41700.0       41700.0       41700       4
1700          0.0  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances    Avg (ns)      Med (ns)     Min (ns)    Max
(ns)   StdDev (ns)                    Name
 --------  ---------------  ---------  ------------  ------------  ----------  ------
----  -----------  ------------------------------------------
    100.0       2468307670          1  2468307670.0  2468307670.0  2468307670  246830
7670          0.0  addVectorsInto(float *, float *, float *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)            Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
     75.5         34159836   2304   14826.3    4382.5      1983     80320      22492.
0  [CUDA Unified Memory memcpy HtoD]
     24.5         11065352    768   14408.0    3743.0      1215     80832      22792.
2  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    402.653   2304     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy HtoD]
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report1.nsys-rep
    /dli/task/report1.sqlite
```

Exercise: Optimize Iteratively

In this exercise you will go through several cycles of editing the execution configuration of 01-vector-add.cu, profiling it, and recording the results to see the impact. Use the following guidelines while working:

- Start by listing 3 to 5 different ways you will update the execution configuration, being sure to cover a range of different grid and block size combinations.

- Edit the 01-vector-add.cu program in one of the ways you listed.
- Compile and profile your updated code with the two code execution cells below.
- Record the runtime of the kernel execution, as given in the profiling output.
- Repeat the edit/profile/record cycle for each possible optimization you listed above

Which of the execution configurations you attempted proved to be the fastest?

In [5]:
```
!nvcc -o iteratively-optimized-vector-add 01-vector-add/01-vector-add.cu -run
```

```
Success! All values calculated correctly.
```

In [6]:
```
!nsys profile --stats=true ./iteratively-optimized-vector-add
```

```
Success! All values calculated correctly.
Generating '/tmp/nsys-report-2294.qdstrm'
[1/8] [========================100%] report3.nsys-rep
[2/8] [========================100%] report3.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report3.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     87.2       1787307628         99  18053612.4  10071669.0      2420  100167994
26409876.0  poll
      9.6        197377683         88   2242928.2   2065294.5       140   20609674
2848749.5  sem_timedwait
      2.1         43065570        497     86651.0     13530.0       410   10532586
601708.1  ioctl
      1.0         20114646         24    838110.3      6290.0       820    7723669
2264055.1  mmap
      0.0           943247         27     34935.1      4940.0      3310     545529
103292.4  mmap64
      0.0           535559         44     12171.8     11205.0      3590      31250
5399.6  open64
      0.0           183634          4     45908.5     42016.0     34251      65351
14875.8  pthread_create
      0.0           181465         29      6257.4      3720.0      1830      36451
7434.3  fopen
      0.0           150591         11     13690.1     13960.0      1250      21250
5249.9  write
      0.0            73100         12      6091.7      3295.0      1200      19530
6356.1  munmap
      0.0            50721         26      1950.8        70.0        60      48961
9588.2  fgets
      0.0            38160          6      6360.0      7295.0      2410       8760
2644.7  open
      0.0            37720         52       725.4       550.0       210       6340
860.2  fcntl
      0.0            28731         22      1306.0      1020.0       550       3810
756.9  fclose
      0.0            24070         14      1719.3      1225.0       380       5230
1393.2  read
      0.0            17551          2      8775.5      8775.5      4260      13291
6385.9  socket
      0.0            11880          1     11880.0     11880.0     11880      11880
0.0  connect
      0.0             9471          5      1894.2      1190.0        60       5911
2397.3  fread
      0.0             7600          1      7600.0      7600.0      7600       7600
0.0  pipe2
      0.0             6500         64       101.6       120.0        40        340
61.5  pthread_mutex_trylock
      0.0             2730          1      2730.0      2730.0      2730       2730
0.0  bind
      0.0             1470          1      1470.0      1470.0      1470       1470
0.0  listen
      0.0              380          1       380.0       380.0       380        380
0.0  pthread_cond_broadcast

[5/8] Executing 'cuda_api_sum' stats report
```

```
 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)          Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     52.6        126797649          3  42265883.0     61242.0     16250  126720157
73139550.2  cudaMallocManaged
     39.1         94180785          1  94180785.0  94180785.0  94180785   94180785
0.0  cudaDeviceSynchronize
      8.4         20143517          3   6714505.7   6520109.0   5850798    7772610
975542.4  cudaFree
      0.0            49660          1     49660.0     49660.0     49660      49660
0.0  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances   Avg (ns)    Med (ns)   Min (ns)  Max (ns)  St
dDev (ns)                    Name
 --------  ---------------  ---------  ----------  ----------  --------  --------  --
---------  ----------------------------------------------
    100.0         94171773          1  94171773.0  94171773.0  94171773  94171773
0.0  addVectorsInto(float *, float *, float *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
     75.3         33687893   2306   14608.8    3103.0      1823     79392      22131.
3  [CUDA Unified Memory memcpy HtoD]
     24.7         11061299    768   14402.7    3727.5      1375     80512      22786.
3  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    402.653   2306     0.175     0.020     0.004     1.036        0.296  [CUDA Unifie
d Memory memcpy HtoD]
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report3.nsys-rep
    /dli/task/report3.sqlite
```

Streaming Multiprocessors and Querying the Device

This section explores how understanding a specific feature of the GPU hardware can promote optimization. After introducing **Streaming Multiprocessors**, you will attempt to further optimize the accelerated vector addition program you have been working on.

The following video presents upcoming material visually, at a high level. Click watch it before moving on to more detailed coverage of their topics in following sections.

In [7]:
```
from IPython.display import HTML

video_url = "https://d36m44n9vdbmda.cloudfront.net/assets/s-ac-04-v1/task2/NVPROF_UM_1

video_html = f"""
<video controls width="640" height="360">
    <source src="{video_url}" type="video/mp4">
    Your browser does not support the video tag.
</video>
"""

display(HTML(video_html))
```

Streaming Multiprocessors and Warps

The GPUs that CUDA applications run on have processing units called **streaming multiprocessors**, or **SMs**. During kernel execution, blocks of threads are given to SMs to execute. In order to support the GPU's ability to perform as many parallel operations as possible, performance gains can often be had by *choosing a grid size that has a number of blocks that is a multiple of the number of SMs on a given GPU.*

Additionally, SMs create, manage, schedule, and execute groupings of 32 threads from within a block called **warps**. A more in depth coverage of SMs and warps is beyond the scope of this course, however, it is important to know that performance gains can also be had by *choosing a block size that has a number of threads that is a multiple of 32.*

Programmatically Querying GPU Device Properties

In order to support portability, since the number of SMs on a GPU can differ depending on the specific GPU being used, the number of SMs should not be hard-coded into a code bases. Rather, this information should be acquired programatically.

The following shows how, in CUDA C/C++, to obtain a C struct which contains many properties about the currently active GPU device, including its number of SMs:

```
int deviceId;
cudaGetDevice(&deviceId);                  // `deviceId` now points to the
id of the currently active GPU.

cudaDeviceProp props;
cudaGetDeviceProperties(&props, deviceId); // `props` now has many useful
properties about
                                           // the active GPU device.
```

Exercise: Query the Device

Currently, 01-get-device-properties.cu contains many unassigned variables, and will print gibberish information intended to describe details about the currently active GPU.

Build out 01-get-device-properties.cu to print the actual values for the desired device properties indicated in the source code. In order to support your work, and as an introduction to them, use the CUDA Runtime Docs to help identify the relevant properties in the device props struct. Refer to the solution if you get stuck.

In [8]:
```
!nvcc -o get-device-properties 04-device-properties/01-get-device-properties.cu -run
```

```
Device ID: 0
Number of SMs: 80
Compute Capability Major: 8
Compute Capability Minor: 6
Warp Size: 32
```

Exercise: Optimize Vector Add with Grids Sized to Number of SMs

Utilize your ability to query the device for its number of SMs to refactor the `addVectorsInto` kernel you have been working on inside 01-vector-add.cu so that it launches with a grid containing a number of blocks that is a multiple of the number of SMs on the device.

Depending on other specific details in the code you have written, this refactor may or may not improve, or significantly change, the performance of your kernel. Therefore, as always, be sure to use `nsys profile` so that you can quantitatively evaluate performance changes. Record the results with the rest of your findings thus far, based on the profiling output.

In [9]:
```
!nvcc -o sm-optimized-vector-add 01-vector-add/01-vector-add.cu -run
```

```
Success! All values calculated correctly.
```

In [10]:
```
!nsys profile --stats=true ./sm-optimized-vector-add
```

```
Success! All values calculated correctly.
Generating '/tmp/nsys-report-1a37.qdstrm'
[1/8] [========================100%] report4.nsys-rep
[2/8] [========================100%] report4.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report4.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     87.9       1787715672         99  18057734.1  10075416.0     27980  100145389
26521155.0  poll
      9.4        190770633         89   2143490.3   2068184.0       180   20378006
2356593.3  sem_timedwait
      1.7         34014918        497     68440.5     11250.0       390    8049623
434584.3  ioctl
      1.0         19643784         24    818491.0      4440.0       890    7177988
2205882.6  mmap
      0.0           887787         27     32881.0      3890.0      3150     544389
103029.7  mmap64
      0.0           506808         44     11518.4     10975.0      3710      30741
4868.8  open64
      0.0           186583         29      6433.9      3990.0      1490      35181
7336.4  fopen
      0.0           185202          4     46300.5     43840.5     33850      63671
14443.8  pthread_create
      0.0           153914         11     13992.2     13891.0     11820      16671
1684.4  write
      0.0            56241         12      4686.8      3460.0      1170      19770
4962.0  munmap
      0.0            50731         26      1951.2        70.0        50      48991
9594.3  fgets
      0.0            42111          6      7018.5      7935.0      3160       9511
2506.8  open
      0.0            35390         52       680.6       495.0       160       6690
883.3  fcntl
      0.0            31970         22      1453.2      1075.0       510       4320
876.1  fclose
      0.0            21690         14      1549.3      1210.0       840       4420
1019.9  read
      0.0            18560          2      9280.0      9280.0      4290      14270
7056.9  socket
      0.0            11070          1     11070.0     11070.0     11070      11070
0.0  connect
      0.0             8571          5      1714.2      1730.0        90       3851
1610.1  fread
      0.0             6850          1      6850.0      6850.0      6850       6850
0.0  pipe2
      0.0             5470         64        85.5        50.0        40        170
47.7  pthread_mutex_trylock
      0.0             2690          1      2690.0      2690.0      2690       2690
0.0  bind
      0.0             1800          1      1800.0      1800.0      1800       1800
0.0  listen
      0.0              280          1       280.0       280.0       280        280
0.0  pthread_cond_broadcast

[5/8] Executing 'cuda_api_sum' stats report
```

```
 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)          Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
---------  ---------------------
     49.1        111303614          3  37101204.7     27550.0     14381  111261683
64224858.5  cudaMallocManaged
     42.1         95381372          1  95381372.0  95381372.0  95381372   95381372
0.0  cudaDeviceSynchronize
      8.7         19695314          3   6565104.7   6631009.0   5852886    7211419
681660.1  cudaFree
      0.0           106891          1    106891.0    106891.0    106891     106891
0.0  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances   Avg (ns)    Med (ns)   Min (ns)  Max (ns)  St
dDev (ns)                    Name
 --------  ---------------  ---------  ----------  ----------  --------  --------  --
---------  ----------------------------------------------
    100.0         95434347          1  95434347.0  95434347.0  95434347  95434347
0.0  addVectorsInto(float *, float *, float *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)             Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
     75.3         33696150   2310   14587.1    3119.0      1823     79615      22123.
8  [CUDA Unified Memory memcpy HtoD]
     24.7         11063310    768   14405.4    3759.5      1343     80736      22790.
2  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    402.653   2310     0.174     0.020     0.004     1.036        0.296  [CUDA Unifie
d Memory memcpy HtoD]
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report4.nsys-rep
    /dli/task/report4.sqlite
```

Unified Memory Details

You have been allocating memory intended for use either by host or device code with `cudaMallocManaged` and up until now have enjoyed the benefits of this method - automatic memory migration, ease of programming - without diving into the details of how the **Unified Memory** (**UM**) allocated by `cudaMallocManaged` actual works.

`nsys profile` provides details about UM management in accelerated applications, and using this information, in conjunction with a more-detailed understanding of how UM works, provides additional opportunities to optimize accelerated applications.

The following video presents upcoming material visually, at a high level. Click watch it before moving on to more detailed coverage of their topics in following sections.

In [11]:
```
from IPython.display import HTML

video_url = "https://d36m44n9vdbmda.cloudfront.net/assets/s-ac-04-v1/task2/NVPROF_UM_2

video_html = f"""
<video controls width="640" height="360">
    <source src="{video_url}" type="video/mp4">
    Your browser does not support the video tag.
</video>
"""

display(HTML(video_html))
```

Unified Memory Migration

When UM is allocated, the memory is not resident yet on either the host or the device. When either the host or device attempts to access the memory, a page fault will occur, at which point the host or device will migrate the needed data in batches. Similarly, at any point when the CPU, or any GPU in the accelerated system, attempts to access memory not yet resident on it, page faults will occur and trigger its migration.

The ability to page fault and migrate memory on demand is tremendously helpful for ease of development in your accelerated applications. Additionally, when working with data that exhibits sparse access patterns, for example when it is impossible to know which data will be required to be worked on until the application actually runs, and for scenarios when data might be accessed by multiple GPU devices in an accelerated system with multiple GPUs, on-demand memory migration is remarkably beneficial.

There are times - for example when data needs are known prior to runtime, and large contiguous blocks of memory are required - when the overhead of page faulting and migrating data on demand incurs an overhead cost that would be better avoided.

Much of the remainder of this lab will be dedicated to understanding on-demand migration, and how to identify it in the profiler's output. With this knowledge you will be able to reduce the overhead of it in scenarios when it would be beneficial.

Exercise: Explore UM Migration and Page Faulting

`nsys profile` provides output describing UM behavior for the profiled application. In this exercise, you will make several modifications to a simple application, and make use of `nsys profile` after each change, to explore how UM data migration behaves.

01-page-faults.cu contains a `hostFunction` and a `gpuKernel`, both which could be used to initialize the elements of a `2<<24` element vector with the number `1`. Currently neither the host function nor GPU kernel are being used.

For each of the 4 questions below, given what you have just learned about UM behavior, first hypothesize about what kind of page faulting should happen, then, edit 01-page-faults.cu to create a scenario, by using one or both of the 2 provided functions in the code bases, that will allow you to test your hypothesis.

In order to test your hypotheses, compile and profile your code using the code execution cells below. Be sure to record your hypotheses, as well as the results, obtained from `nsys profile --stats=true` output. In the output of `nsys profile --stats=true` you should be looking for the following:

- Is there a *CUDA Memory Operation Statistics* section in the output?
- If so, does it indicate host to device (HtoD) or device to host (DtoH) migrations?
- When there are migrations, what does the output say about how many *Operations* there were? If you see many small memory migration operations, this is a sign that on-demand page faulting is occurring, with small memory migrations occurring each time there is a page fault in the requested location.

Here are the scenarios for you to explore, along with solutions for them if you get stuck:

- Is there evidence of memory migration and/or page faulting when unified memory is accessed only by the CPU? (solution)

- Is there evidence of memory migration and/or page faulting when unified memory is accessed only by the GPU? (solution)
- Is there evidence of memory migration and/or page faulting when unified memory is accessed first by the CPU then the GPU? (solution)
- Is there evidence of memory migration and/or page faulting when unified memory is accessed first by the GPU then the CPU? (solution)

In [18]:
```
!nvcc -o page-faults 06-unified-memory-page-faults/01-page-faults.cu -run
```

In [19]:
```
!nsys profile --stats=true ./page-faults
```

```
Generating '/tmp/nsys-report-a9f2.qdstrm'
[1/8] [========================100%] report8.nsys-rep
[2/8] [========================100%] report8.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report8.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     80.6        541726040         38  14255948.4  10068452.0     27571  100140983
21705239.3  poll
     12.2         82281413         34   2420041.6   2064009.0       120   20479630
4309413.5  sem_timedwait
      5.7         38007919        483     78691.3     13460.0       400    9256664
522009.6  ioctl
      1.1          7601187         18    422288.2      4670.0      1140    7471044
1759163.9  mmap
      0.2          1133120         27     41967.4      4200.0      3380     739722
140327.5  mmap64
      0.1           525306         44     11938.8     10735.0      4690      31891
5149.4  open64
      0.0           192682          4     48170.5     47510.5     37430      60231
12245.9  pthread_create
      0.0           190955         29      6584.7      3820.0      1490      44471
8426.2  fopen
      0.0           146241         11     13294.6     13770.0      2770      20580
4986.8  write
      0.0            58380         26      2245.4        90.0        70      56230
11010.7  fgets
      0.0            48210          7      6887.1      4720.0      3240      20150
5987.7  munmap
      0.0            40840          6      6806.7      6905.0      3480       9660
2362.4  open
      0.0            35351         52       679.8       545.0       160       5550
731.7  fcntl
      0.0            32282         22      1467.4      1235.0       750       3880
737.3  fclose
      0.0            20390         14      1456.4      1155.0       500       3880
1040.0  read
      0.0            17191          2      8595.5      8595.5      3770      13421
6824.3  socket
      0.0            14090          1     14090.0     14090.0     14090      14090
0.0  connect
      0.0             9110          5      1822.0      1490.0        90       3700
1775.2  fread
      0.0             5981          1      5981.0      5981.0      5981       5981
0.0  pipe2
      0.0             5590         64        87.3        50.0        40        480
67.7  pthread_mutex_trylock
      0.0             2220          1      2220.0      2220.0      2220       2220
0.0  bind
      0.0             1540          1      1540.0      1540.0      1540       1540
0.0  listen
      0.0              150          1       150.0       150.0       150        150
0.0  pthread_cond_broadcast

[5/8] Executing 'cuda_api_sum' stats report
```

```
 Time (%)  Total Time (ns)  Num Calls   Avg (ns)     Med (ns)    Min (ns)   Max (ns)
StdDev (ns)          Name
 --------  ---------------  ---------  -----------  -----------  ---------  ---------
-----------  ---------------------
     83.9        116120559          1  116120559.0  116120559.0  116120559  116120559
0.0  cudaMallocManaged
     10.6         14605032          1   14605032.0   14605032.0   14605032   14605032
0.0  cudaDeviceSynchronize
      5.5          7578756          1    7578756.0    7578756.0    7578756    7578756
0.0  cudaFree
      0.0            44471          1      44471.0      44471.0      44471      44471
0.0  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances   Avg (ns)    Med (ns)    Min (ns)  Max (ns)  St
dDev (ns)          Name
 --------  ---------------  ---------  ----------  ----------  --------  --------  --
---------  ------------------------
    100.0         14602092          1  14602092.0  14602092.0  14602092  14602092
0.0  deviceKernel(int *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
    100.0         11082131    768   14429.9    3759.5      1439     80768      22782.
7  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report8.nsys-rep
    /dli/task/report8.sqlite
```

Exercise: Revisit UM Behavior for Vector Add Program

Returning to the 01-vector-add.cu program you have been working on throughout this lab, review the code bases in its current state, and hypothesize about what kinds of memory migrations and/or page faults you expect to occur. Look at the profiling output for your last refactor (either by scrolling up to find the output or by executing the code execution cell just below), observing the *CUDA Memory Operation Statistics* section of the profiler output. Can you explain the kinds of migrations and the number of their operations based on the contents of the code base?

In [20]:
```
!nsys profile --stats=true ./sm-optimized-vector-add
```

```
Success! All values calculated correctly.
Generating '/tmp/nsys-report-d526.qdstrm'
[1/8] [========================100%] report9.nsys-rep
[2/8] [========================100%] report9.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report9.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     87.4       1786831510         99  18048803.1  10071688.0      2640  100138764
26450462.4  poll
      9.4        191500807         88   2176145.5   2066619.0       110   20422889
2536213.2  sem_timedwait
      1.9         39179211        497     78831.4     12610.0       380    8812336
497030.3  ioctl
      1.2         23651322         24    985471.8      4370.0       930   10332771
2728468.9  mmap
      0.1          1094176         27     40525.0      3820.0      2870     752582
142899.8  mmap64
      0.0           475821         44     10814.1     10405.5      4270      29420
4122.1  open64
      0.0           193833          4     48458.3     46325.5     37051      64131
12854.3  pthread_create
      0.0           192775         29      6647.4      4160.0      1410      32561
7338.2  fopen
      0.0           143232         11     13021.1     13210.0      1050      22750
5892.0  write
      0.0            59582         26      2291.6        90.0        70      57411
11242.2  fgets
      0.0            56441         11      5131.0      4790.0      1090      16760
4162.6  munmap
      0.0            53060         22      2411.8      1505.0       750      19900
3985.0  fclose
      0.0            43961          6      7326.8      8135.0      3521      10020
2592.8  open
      0.0            37171         52       714.8       470.0       150       6810
921.9  fcntl
      0.0            20710         14      1479.3      1105.0       420       4040
1182.2  read
      0.0            16850          2      8425.0      8425.0      4050      12800
6187.2  socket
      0.0            12100          1     12100.0     12100.0     12100      12100
0.0  connect
      0.0             9871          5      1974.2      1420.0        90       4961
2032.2  fread
      0.0             7700          1      7700.0      7700.0      7700       7700
0.0  pipe2
      0.0             6740         64       105.3       130.0        50        270
50.3  pthread_mutex_trylock
      0.0             2210          1      2210.0      2210.0      2210       2210
0.0  bind
      0.0             1241          1      1241.0      1241.0      1241       1241
0.0  listen
      0.0              270          1       270.0       270.0       270        270
0.0  pthread_cond_broadcast

[5/8] Executing 'cuda_api_sum' stats report
```

```
 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)          Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     49.7        117268209          3  39089403.0     24171.0     13340  117230698  
67672346.8  cudaMallocManaged
     40.3         95055879          1  95055879.0  95055879.0  95055879   95055879  
0.0  cudaDeviceSynchronize
     10.1         23774695          3   7924898.3   7255521.0   6093601   10425573  
2242218.7  cudaFree
      0.0            45431          1     45431.0     45431.0     45431      45431  
0.0  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances   Avg (ns)    Med (ns)   Min (ns)  Max (ns)  St
dDev (ns)                   Name
 --------  ---------------  ---------  ----------  ----------  --------  --------  --
---------  ----------------------------------------------
    100.0         95047152          1  95047152.0  95047152.0  95047152  95047152  
0.0  addVectorsInto(float *, float *, float *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
     75.3         33718457   2322   14521.3    2847.0      1822     79424      22092.
3  [CUDA Unified Memory memcpy HtoD]
     24.7         11068084    768   14411.6    3727.5      1343     80607      22789.
8  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)  
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    402.653   2322     0.173     0.012     0.004     1.036        0.296  [CUDA Unifie
d Memory memcpy HtoD]
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report9.nsys-rep
    /dli/task/report9.sqlite
```

Exercise: Initialize Vector in Kernel

When `nsys profile` gives the amount of time that a kernel takes to execute, the host-to-device page faults and data migrations that occur during this kernel's execution are included in the displayed execution time.

With this in mind, refactor the `initWith` host function in your 01-vector-add.cu program to instead be a CUDA kernel, initializing the allocated vector in parallel on the GPU. After successfully compiling and running the refactored application, but before profiling it, hypothesize about the following:

- How do you expect the refactor to affect UM memory migration behavior?
- How do you expect the refactor to affect the reported run time of `addVectorsInto`?

Once again, record the results. Refer to the solution if you get stuck.

In [21]:
```
!nvcc -o initialize-in-kernel 01-vector-add/01-vector-add.cu -run
```

```
Device ID: 0    Number of SMs: 80
Success! All values calculated correctly.
```

In [22]:
```
!nsys profile --stats=true ./initialize-in-kernel
```

```
Device ID: 0    Number of SMs: 80
Success! All values calculated correctly.
Generating '/tmp/nsys-report-cece.qdstrm'
[1/8] [========================100%] report10.nsys-rep
[2/8] [========================100%] report10.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report10.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ----------------------
     80.4        491524714         33  14894688.3  10068707.0      2040  100129715  
23316814.7  poll
     10.9         66394903         29   2289479.4   2064834.0       210   20431919  
4101101.0  sem_timedwait
      5.8         35429236        497     71286.2     10660.0       390    8054254  
474155.6  ioctl
      2.6         15882726         24    661780.3      5280.0       910    7235021  
1843539.9  mmap
      0.1           903494         27     33462.7      3830.0      2920     559689  
106096.4  mmap64
      0.1           461570         44     10490.2      9780.0      3430      23820  
3902.3  open64
      0.0           166323          4     41580.8     41585.5     33561      49591  
6901.5  pthread_create
      0.0           149143         29      5142.9      3880.0      1710      20591  
4554.2  fopen
      0.0           138292         11     12572.0     13870.0      1090      16850  
4599.5  write
      0.0            60191         26      2315.0        70.0        60      58301  
11419.0  fgets
      0.0            49300         11      4481.8      3490.0      1190      19150  
4986.3  munmap
      0.0            34720         52       667.7       460.0       160       5410  
756.5  fcntl
      0.0            32230          6      5371.7      5005.0      2400       8830  
2301.2  open
      0.0            26010         22      1182.3      1075.0       530       3330  
639.5  fclose
      0.0            21601         14      1542.9      1290.0       880       3330  
764.1  read
      0.0            17031          5      3406.2      1640.0       100       9360  
3733.8  fread
      0.0            11270          2      5635.0      5635.0      3760       7510  
2651.7  socket
      0.0             7871          1      7871.0      7871.0      7871       7871  
0.0  connect
      0.0             6631          1      6631.0      6631.0      6631       6631  
0.0  pipe2
      0.0             5500         64        85.9        50.0        40        190  
46.9  pthread_mutex_trylock
      0.0             2040          1      2040.0      2040.0      2040       2040  
0.0  bind
      0.0             1270          1      1270.0      1270.0      1270       1270  
0.0  listen
      0.0              310          1       310.0       310.0       310        310  
0.0  pthread_cond_broadcast
```

```
[5/8] Executing 'cuda_api_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)          Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     71.6        120479863          3  40159954.3     52001.0     14260  120413602  
69501700.2  cudaMallocManaged
     19.0         31936600          2  15968300.0  15968300.0    859804   31076796  
21366640.0  cudaDeviceSynchronize
      9.4         15906725          3   5302241.7   4368363.0   4276671    7261691  
1697552.1  cudaFree
      0.0            60671          4     15167.8     12485.0      4291      31410  
12593.2  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances   Avg (ns)    Med (ns)   Min (ns)  Max (ns)  St
dDev (ns)                     Name
 --------  ---------------  ---------  ----------  ----------  --------  --------  --
---------  --------------------------------------------
     97.3         31079831          3  10359943.7  10519954.0   9974579  10585298  
335331.0  initWith(float, float *, int)
      2.7           856991          1    856991.0    856991.0    856991    856991  
0.0  addArraysInto(float *, float *, float *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
    100.0         11079729    768   14426.7    3791.5      1439     80736      22783.
6  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)  
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report10.nsys-rep
    /dli/task/report10.sqlite
```

Asynchronous Memory Prefetching

A powerful technique to reduce the overhead of page faulting and on-demand memory migrations, both in host-to-device and device-to-host memory transfers, is called **asynchronous memory prefetching**. Using this technique allows programmers to asynchronously migrate unified memory (UM) to any CPU or GPU device in the system, in the background, prior to its use by application code. By doing this, GPU kernels and CPU function performance can be increased on account of reduced page fault and on-demand data migration overhead.

Prefetching also tends to migrate data in larger chunks, and therefore fewer trips, than on-demand migration. This makes it an excellent fit when data access needs are known before runtime, and when data access patterns are not sparse.

CUDA Makes asynchronously prefetching managed memory to either a GPU device or the CPU easy with its `cudaMemPrefetchAsync` function. Here is an example of using it to both prefetch data to the currently active GPU device, and then, to the CPU:

```
int deviceId;
cudaGetDevice(&deviceId);                                         // The ID
of the currently active GPU device.

cudaMemPrefetchAsync(pointerToSomeUMData, size, deviceId);        //
Prefetch to GPU device.
cudaMemPrefetchAsync(pointerToSomeUMData, size, cudaCpuDeviceId); //
Prefetch to host. `cudaCpuDeviceId` is a
                                                                  // built-
in CUDA variable.
```

Exercise: Prefetch Memory

At this point in the lab, your 01-vector-add.cu program should not only be launching a CUDA kernel to add 2 vectors into a third solution vector, all which are allocated with `cudaMallocManaged`, but should also be initializing each of the 3 vectors in parallel in a CUDA kernel. If for some reason, your application does not do any of the above, please refer to the following reference application, and update your own code bases to reflect its current functionality.

Conduct 3 experiments using `cudaMemPrefetchAsync` inside of your 01-vector-add.cu application to understand its impact on page-faulting and memory migration.

- What happens when you prefetch one of the initialized vectors to the device?
- What happens when you prefetch two of the initialized vectors to the device?
- What happens when you prefetch all three of the initialized vectors to the device?

Hypothesize about UM behavior, page faulting specifically, as well as the impact on the reported run time of the initialization kernel, before each experiment, and then verify by running `nsys profile`. Refer to the solution if you get stuck.

In [23]:
```
!nvcc -o prefetch-to-gpu 01-vector-add/01-vector-add.cu -run
```

```
Success! All values calculated correctly.
```

In [24]:
```
!nsys profile --stats=true ./prefetch-to-gpu
```

```
Success! All values calculated correctly.
Generating '/tmp/nsys-report-4331.qdstrm'
[1/8] [========================100%] report11.nsys-rep
[2/8] [========================100%] report11.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report11.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     73.6        361104340         29  12451873.8  10066695.0      2500  100130275
18396348.4  poll
     12.3         60417894         26   2323765.2   2060264.0       120   20471206
4221826.1  sem_timedwait
     10.4         51054033        500    102108.1     13980.0       400   13713676
748892.5  ioctl
      3.2         15574635         24    648943.1      4445.0      1000    7180498
1812329.2  mmap
      0.2          1114249         27     41268.5      4391.0      3490     744972
141381.4  mmap64
      0.1           531899         44     12088.6     11595.0      5370      30771
4874.0  open64
      0.0           197153          4     49288.3     47966.0     37860      63361
11429.1  pthread_create
      0.0           188491         29      6499.7      4570.0      1860      28960
5975.2  fopen
      0.0           147504         11     13409.5     15570.0      1150      17851
4675.3  write
      0.0            59471         26      2287.3        90.0        70      57181
11196.2  fgets
      0.0            51001          6      8500.2      8125.5      3870      15470
4196.0  open
      0.0            41721         12      3476.8      2945.5      1420       7620
1757.1  munmap
      0.0            38100         52       732.7       540.0       160       6240
853.5  fcntl
      0.0            35541         22      1615.5      1495.0       750       3400
767.6  fclose
      0.0            20711         14      1479.4      1185.0       360       4481
1278.9  read
      0.0            18030          5      3606.0      1810.0        90      10040
3965.3  fread
      0.0            15970          2      7985.0      7985.0      4210      11760
5338.7  socket
      0.0            11450          1     11450.0     11450.0     11450      11450
0.0  connect
      0.0             6930          1      6930.0      6930.0      6930       6930
0.0  pipe2
      0.0             5550         64        86.7        50.0        40        180
46.3  pthread_mutex_trylock
      0.0             2720          1      2720.0      2720.0      2720       2720
0.0  bind
      0.0             1380          1      1380.0      1380.0      1380       1380
0.0  listen
      0.0              280          1       280.0       280.0       280        280
0.0  pthread_cond_broadcast
      0.0              190          1       190.0       190.0       190        190
0.0  pthread_mutex_lock
```

```
[5/8] Executing 'cuda_api_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)   Med (ns)   Min (ns)  Max (ns)   St
dDev (ns)          Name
 --------  ---------------  ---------  ----------  ---------  --------  ---------  --
---------  ---------------------
     87.2        128099253          3  42699751.0    29151.0     13870  128056232   7
3920881.3  cudaMallocManaged
     10.6         15568305          3   5189435.0  4225669.0   4137478    7205158
1746224.2  cudaFree
      1.1          1687817          1   1687817.0  1687817.0   1687817    1687817
0.0  cudaDeviceSynchronize
      1.1          1580046          3    526682.0   531579.0    498928     549539
25658.4  cudaMemPrefetchAsync
      0.0            37881          4      9470.3     5275.5      4280      23050
9080.2  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev
(ns)                      Name
 --------  ---------------  ---------  --------  --------  --------  --------  ------
-----  ---------------------------------------------
     50.1           850879          1  850879.0  850879.0    850879    850879
0.0  addVectorsInto(float *, float *, float *, int)
     49.9           847678          3  282559.3  283679.0    280127    283872       2
108.7  initWith(float, float *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)            Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
    100.0         11074504    768   14419.9    3791.5      1439     80640      22780.
4  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report11.nsys-rep
    /dli/task/report11.sqlite
```

Exercise: Prefetch Memory Back to the CPU

Add additional prefetching back to the CPU for the function that verifies the correctness of the `addVectorInto` kernel. Again, hypothesize about the impact on UM before profiling in `nsys` to confirm. Refer to the solution if you get stuck.

In [25]:
```
!nvcc -o prefetch-to-cpu 01-vector-add/01-vector-add.cu -run
```

```
Success! All values calculated correctly.
```

In [26]:
```
!nsys profile --stats=true ./prefetch-to-cpu
```

```
Success! All values calculated correctly.
Generating '/tmp/nsys-report-28d8.qdstrm'
[1/8] [========================100%] report12.nsys-rep
[2/8] [========================100%] report12.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report12.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   S
tdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -
----------  ---------------------
     74.3        360997711         29  12448196.9  10071696.0      2470  100128328
18660246.8  poll
     12.1         58798296         26   2261472.9   2065228.5       300   20539378
4254529.3  sem_timedwait
      9.8         47851040        500     95702.1     13090.5       400    9951564
621877.0  ioctl
      3.2         15651428         24    652142.8      6930.0      1100    7233319
1821732.4  mmap
      0.2          1180200         27     43711.1      4290.0      2920     805524
152984.9  mmap64
      0.1           524428         44     11918.8     10850.5      4900      36191
5720.4  open64
      0.0           205142         29      7073.9      3650.0      1440      47760
9487.4  fopen
      0.0           176753          4     44188.3     46500.5     26481      57271
14762.6  pthread_create
      0.0           143303         11     13027.5     12600.0      1220      20661
5249.0  write
      0.0            77863         11      7078.5      4510.0      1940      34971
9413.4  munmap
      0.0            57841         26      2224.7        90.0        70      55601
10886.7  fgets
      0.0            46432          6      7738.7      8705.5      3310      10120
2741.0  open
      0.0            37910         52       729.0       540.0       150       6800
905.2  fcntl
      0.0            33400         22      1518.2      1275.0       750       4800
873.9  fclose
      0.0            21220         14      1515.7      1260.0       450       4500
1197.9  read
      0.0            12661          5      2532.2      1790.0        80       6011
2246.4  fread
      0.0            11920          2      5960.0      5960.0      4090       7830
2644.6  socket
      0.0             9540          1      9540.0      9540.0      9540       9540
0.0  connect
      0.0             7060          1      7060.0      7060.0      7060       7060
0.0  pipe2
      0.0             6080         64        95.0        50.0        40        430
66.3  pthread_mutex_trylock
      0.0             2450          1      2450.0      2450.0      2450       2450
0.0  listen
      0.0             2160          1      2160.0      2160.0      2160       2160
0.0  bind
      0.0              310          1       310.0       310.0       310        310
0.0  pthread_cond_broadcast

[5/8] Executing 'cuda_api_sum' stats report
```

```
 Time (%)  Total Time (ns)  Num Calls   Avg (ns)   Med (ns)   Min (ns)  Max (ns)   St
dDev (ns)         Name
 --------  ---------------  ---------  ----------  ---------  --------  ---------  --
---------  ---------------------
     86.7        125742849          3  41914283.0    70801.0     31130  125640918   7
2509395.6  cudaMallocManaged
     10.8         15664497          3   5221499.0  4257210.0   4148718    7258569
1764988.2  cudaFree
      1.3          1917881          3    639293.7   526649.0    517588     873644
203003.9  cudaMemPrefetchAsync
      1.2          1696098          1   1696098.0  1696098.0   1696098    1696098
0.0  cudaDeviceSynchronize
      0.0            46221          4     11555.3     5695.0      4240      30591
12709.9  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev
(ns)                       Name
 --------  ---------------  ---------  --------  --------  --------  --------  ------
-----  ----------------------------------------------
     50.5           863806          1  863806.0  863806.0    863806    863806
0.0  addVectorsInto(float *, float *, float *, int)
     49.5           845023          3  281674.3  282719.0    279552    282752       1
838.1  initWith(float, float *, int)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
    100.0         11082574    768   14430.4    3775.5      1439     80768      22792.
2  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
    134.218    768     0.175     0.033     0.004     1.044        0.301  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report12.nsys-rep
    /dli/task/report12.sqlite
```

After this series of refactors to use asynchronous prefetching, you should see that there are fewer, but larger, memory transfers, and, that the kernel execution time is significantly decreased.

Summary

At this point in the lab, you are able to:

- Use the Nsight Systems command line tool (**nsys**) to profile accelerated application performance.
- Leverage an understanding of **Streaming Multiprocessors** to optimize execution configurations.
- Understand the behavior of **Unified Memory** with regard to page faulting and data migrations.
- Use **asynchronous memory prefetching** to reduce page faults and data migrations for increased performance.
- Employ an iterative development cycle to rapidly accelerate and deploy applications.

In order to consolidate your learning, and reinforce your ability to iteratively accelerate, optimize, and deploy applications, please proceed to this lab's final exercise. After completing it, for those of you with time and interest, please proceed to the *Advanced Content* section.

Final Exercise: Iteratively Optimize an Accelerated SAXPY Application

A basic accelerated SAXPY (Single Precision a*x+b) application has been provided for you here. It currently works and you can compile, run, and then profile it with `nsys profile` below.

Record the runtime of the `saxpy` kernel without making any modifications and then work *iteratively* to optimize the application, using `nsys profile` after each iteration to notice the effects of the code changes on kernel performance and UM behavior.

Utilize the techniques from this lab. To support your learning, utilize effortful retrieval whenever possible, rather than rushing to look up the specifics of techniques from earlier in the lesson.

Your end goal is to profile an accurate `saxpy` kernel, without modifying `N`, to run in under *200,000 ns*. Check out the solution if you get stuck, and feel free to compile and profile it if you wish.

In [27]:
```
!nvcc -o saxpy 09-saxpy/01-saxpy.cu -run
```

```
c[0] = 5, c[1] = 5, c[2] = 5, c[3] = 5, c[4] = 5,
c[4194299] = 5, c[4194300] = 5, c[4194301] = 5, c[4194302] = 5, c[4194303] = 5,
```

In [28]:
```
!nsys profile --stats=true ./saxpy
```

```
c[0] = 5, c[1] = 5, c[2] = 5, c[3] = 5, c[4] = 5,
c[4194299] = 5, c[4194300] = 5, c[4194301] = 5, c[4194302] = 5, c[4194303] = 5,
Generating '/tmp/nsys-report-6f45.qdstrm'
[1/8] [========================100%] report13.nsys-rep
[2/8] [========================100%] report13.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /dli/task/report13.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)  St
dDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  --------  --
---------  ----------------------
     71.9        291987274         26  11230279.8  10068171.0      2140  96765905   1
9297595.6  poll
     14.8         59922913        500    119845.8     13410.0       440  14182854
793294.2  ioctl
     12.0         48810974         19   2568998.6     59691.0       140  20476827
5511928.7  sem_timedwait
      0.6          2294798         23     99773.8      6860.0      1160    754462
244946.9  mmap
      0.3          1161698         27     43025.9      4250.0      3320    760462
144332.4  mmap64
      0.1           599700          3    199900.0    209523.0     96942    293235
98499.7  sem_wait
      0.1           535706         44     12175.1     11720.0      4840     34551
5042.9  open64
      0.1           264804          5     52960.8     47950.0     39911     77811
15848.6  pthread_create
      0.0           190326         29      6563.0      4110.0      1671     34110
6760.4  fopen
      0.0           166595         13     12815.0     12930.0      1140     19011
4519.6  write
      0.0            60171         26      2314.3        90.0        70     57931
11343.6  fgets
      0.0            47250          6      7875.0      8245.0      4200     10700
2592.4  open
      0.0            36721         52       706.2       515.0       220      6330
860.3  fcntl
      0.0            35310         22      1605.0      1435.0       770      3890
740.0  fclose
      0.0            34630          9      3847.8      3660.0      1490      6710
1734.9  munmap
      0.0            26080         16      1630.0      1300.0       570      4950
1222.9  read
      0.0            18660          2      9330.0      9330.0      4390     14270
6986.2  socket
      0.0            12971          1     12971.0     12971.0     12971     12971
0.0  connect
      0.0            12600          5      2520.0      1420.0        90      7620
3135.2  fread
      0.0             7890          1      7890.0      7890.0      7890      7890
0.0  pipe2
      0.0             5680         64        88.8        80.0        40       180
46.0  pthread_mutex_trylock
      0.0             2500          1      2500.0      2500.0      2500      2500
0.0  bind
      0.0             1210          1      1210.0      1210.0      1210      1210
0.0  listen
      0.0              380          1       380.0       380.0       380       380
```

```
0.0  pthread_cond_broadcast

[5/8] Executing 'cuda_api_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)   Med (ns)   Min (ns)  Max (ns)   St
dDev (ns)          Name
 --------  ---------------  ---------  ----------  ---------  --------  ---------  --
---------  ---------------------
     94.7        136303807          3  45434602.3    24350.0     18550  136260907   7
8657887.2  cudaMallocManaged
      2.6          3797253          1   3797253.0  3797253.0   3797253    3797253
0.0  cudaDeviceSynchronize
      1.6          2252198          3    750732.7   743992.0    710762     797444
43732.4  cudaFree
      1.0          1474414          3    491471.3   212223.0      7900    1254291
668473.9  cudaMemPrefetchAsync
      0.0            36321          1     36321.0    36321.0     36321      36321
0.0  cudaLaunchKernel

[6/8] Executing 'cuda_gpu_kern_sum' stats report

 Time (%)  Total Time (ns)  Instances  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev
(ns)            Name
 --------  ---------------  ---------  --------  --------  --------  --------  ------
-----  -------------------------
    100.0           105824          1  105824.0  105824.0    105824    105824
0.0  saxpy(int *, int *, int *)

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (n
s)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  ----------
-  ---------------------------------
     99.6          3813656     24  158902.3  158784.0    158656    159232        230.
8  [CUDA Unified Memory memcpy HtoD]
      0.4            14398      4    3599.5    3583.5      1407      5824       2459.
0  [CUDA Unified Memory memcpy DtoH]

[8/8] Executing 'cuda_gpu_mem_size_sum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)
Operation
 ----------  -----  --------  --------  --------  --------  -----------  ------------
---------------------
     50.332     24     2.097     2.097     2.097     2.097        0.000  [CUDA Unifie
d Memory memcpy HtoD]
      0.131      4     0.033     0.033     0.004     0.061        0.033  [CUDA Unifie
d Memory memcpy DtoH]

Generated:
    /dli/task/report13.nsys-rep
    /dli/task/report13.sqlite
```

In [ ]: