Introduction to NVIDIA Profilers

Green AI 2020: Tutorial and Hackathon

Tuesday, January 28, 2020

Source: https://www.olcf.ornl.gov/wp-content/uploads/2019/04/Intro_to_NVIDIA_profilers.pdf

Profiling on NVIDIA GPUs

- NVIDIA Profiler: nvprof
- NVIDIA Visual Profiler: nvvp

- Satori has nvprof installed all nodes
  - Permissions are being updated for users on all nodes (~25% nodes complete)

- Outline
  - CUDA Vector Addition Example
    - Profile and Analyze via nvprof text-based output and NVIDIA Visual Profiler
  - TensorFlow and PyTorch Profiling
    - NVIDIA APEX: Kernel-Specific Profiling in PyTorch
  - Directional Resources

CUDA Example: vector_add.cu

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Vector Addition Kernel (GPU)

```
__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}
```

CUDA Example: vector_add.cu

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Allocate Memory on CPU

```
int *A = (int *) malloc(bytes);
int *B = (int *) malloc(bytes);
int *C = (int *) malloc(bytes);
int *d_A, *d_B, *d_C;
```

CUDA Example: vector_add.cu

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Allocate Memory on GPU

```
cudaMalloc(&d_A, bytes);
cudaMalloc(&d_B, bytes);
cudaMalloc(&d_C, bytes);
```

CUDA Example: vector_add.cu

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Initialize Arrays on CPU

```
for(int i=0; i<N; i++){
    A[i] = 1;
    B[i] = 2;
}
```

CUDA Example: vector_add.cu

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Copy Data From CPU to GPU

```
cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);
```

CUDA Example: vector_add.cu

```cpp
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Set Configuration Parameters and Launch Kernel

```cpp
int thr_per_blk = 256;
int blk_in_grid = ceil( float(N) / thr_per_blk );
add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);
```

CUDA Example: vector_add.cu

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Copy Result From GPU to CPU

```
cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);
```

CUDA Example: vector_add.cu

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

Free Memory on CPU and GPU

```
free(A);
free(B);
free(C);
cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
```

CUDA Example: vector_add.cu

- **Build**

  $ nvcc vector_add.cu –o vector_add

- **Execute and Profile**

  [user@service0001] $ bsub -gpu "num=1" -Is bash

  [user@node00XX] $ nvprof –s –o results.nvvp ./vector_add

Invoke the command line NVIDIA profiler (nvprof)

-s: Print summary of profiling results
(text-based; default unless –o used)

-o: Save profiling timeline results to file
(used with NVIDIA Visual Profiler)

CUDA Example: vector_add.cu

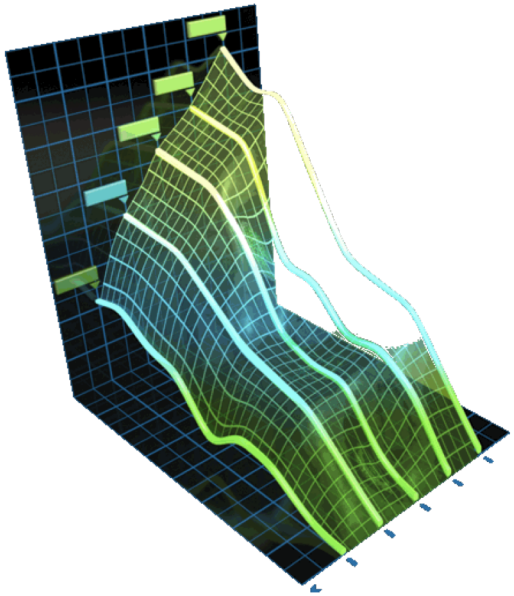
- **nvprof results (text only)**

```
==151936== NVPROF is profiling process 151936, command: ./add_vectors
==151936== Profiling application: ./add_vectors
==151936== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   50.63%  465.79us         2  232.90us  225.95us  239.84us  [CUDA memcpy HtoD]
                   47.04%  432.77us         1  432.77us  432.77us  432.77us  [CUDA memcpy DtoH]
                    2.33%  21.440us         1  21.440us  21.440us  21.440us  add_vectors(int*, int*, int*)
      API calls:   96.37%  280.56ms         3  93.518ms  840.70us  278.84ms  cudaMalloc
                    1.35%  3.9208ms         2  1.9604ms  1.9547ms  1.9661ms  cuDeviceTotalMem
                    1.03%  3.0079ms       194  15.504us     289ns  621.78us  cuDeviceGetAttribute
                    0.66%  1.9331ms         3  644.36us  623.20us  686.45us  cudaFree
                    0.47%  1.3773ms         3  459.10us  296.53us  735.09us  cudaMemcpy
                    0.09%  248.69us         2  124.35us  122.61us  126.08us  cuDeviceGetName
                    0.02%  63.962us         1  63.962us  63.962us  63.962us  cudaLaunchKernel
                    0.00%  7.8380us         2  3.9190us  3.8130us  4.0250us  cuDeviceGetPCIBusId
                    0.00%  2.6260us         4     656ns     402ns  1.1760us  cuDeviceGet
                    0.00%  1.9210us         3     640ns     441ns     830ns  cuDeviceGetCount
                    0.00%  1.0150us         2     507ns     429ns     586ns  cuDeviceGetUuid
```

CUDA Example: vector_add.cu

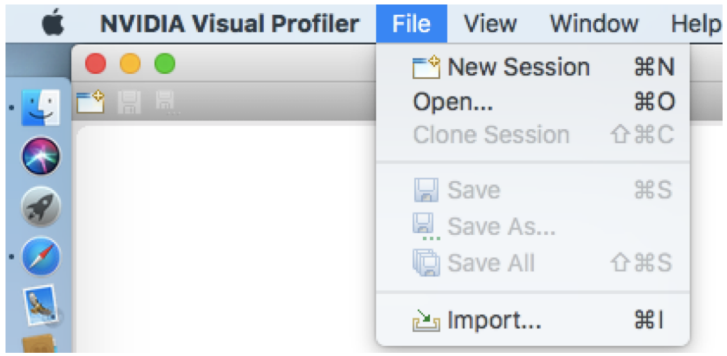
- Install CUDA Toolkit on local machine (includes NVIDIA Visual Profiler [nvvp])
- Transfer results.nvvp to your local machine to view in NVIDIA Visual Profiler

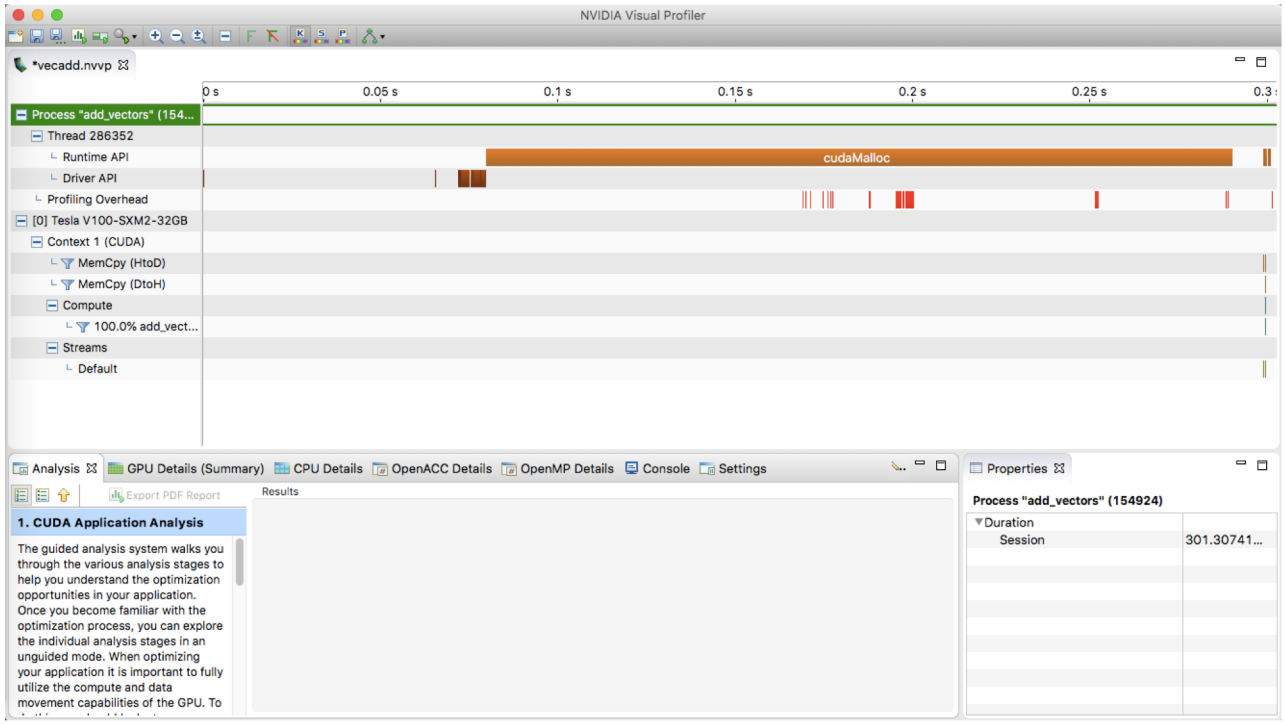
```
[local] $ scp USERNAME@satori-login-001.mit.edu:/path/to/results.nvvp    /path/to/local/dest
```



CUDA Example: vector_add.cu

① Load Data – Open… *.nvvp



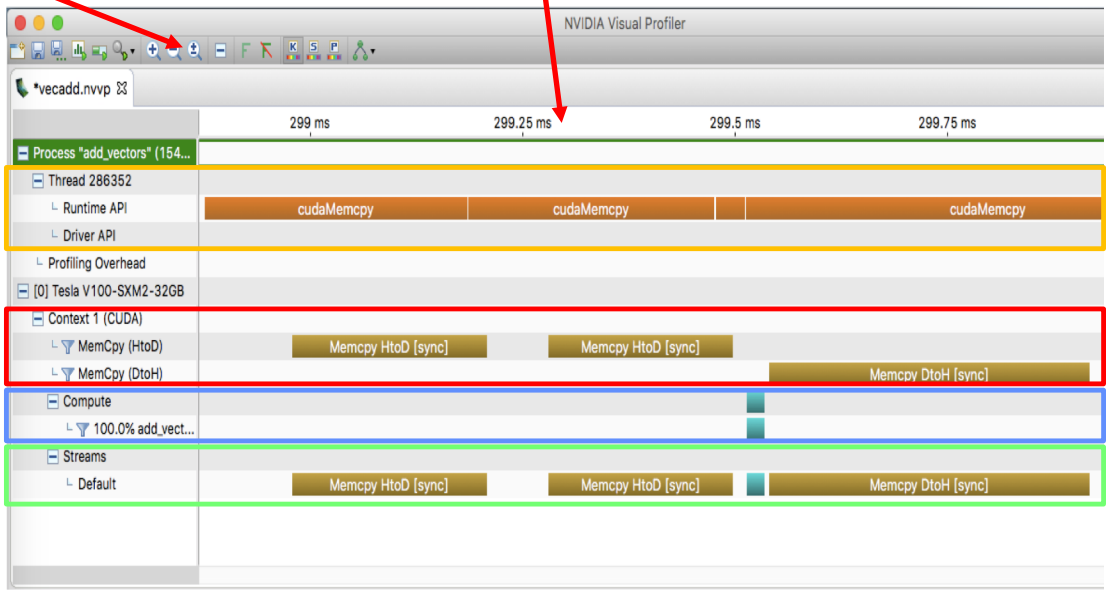


CUDA Example: vector_add.cu

2 To view specific region, hold Ctrl + left-click and drag mouse over region (OSX: Command)

Reset view

Left-click timeline and drag mouse to measure specific regions



Kernel Launches

Memory Ops

Kernel Execution

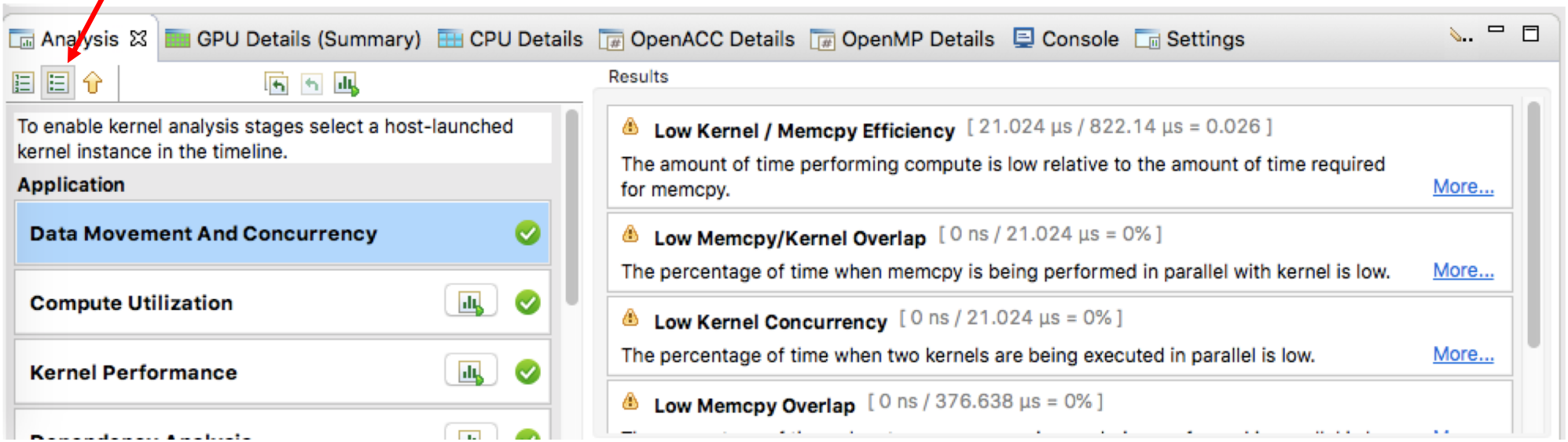
GPU Streams

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int *) malloc(bytes);
    int *B = (int *) malloc(bytes);
    int *C = (int *) malloc(bytes);
    int *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

NVPROF: Unguided Analysis

Switch To Unguided Analysis

Provides information regarding GPU Utilization and Performance



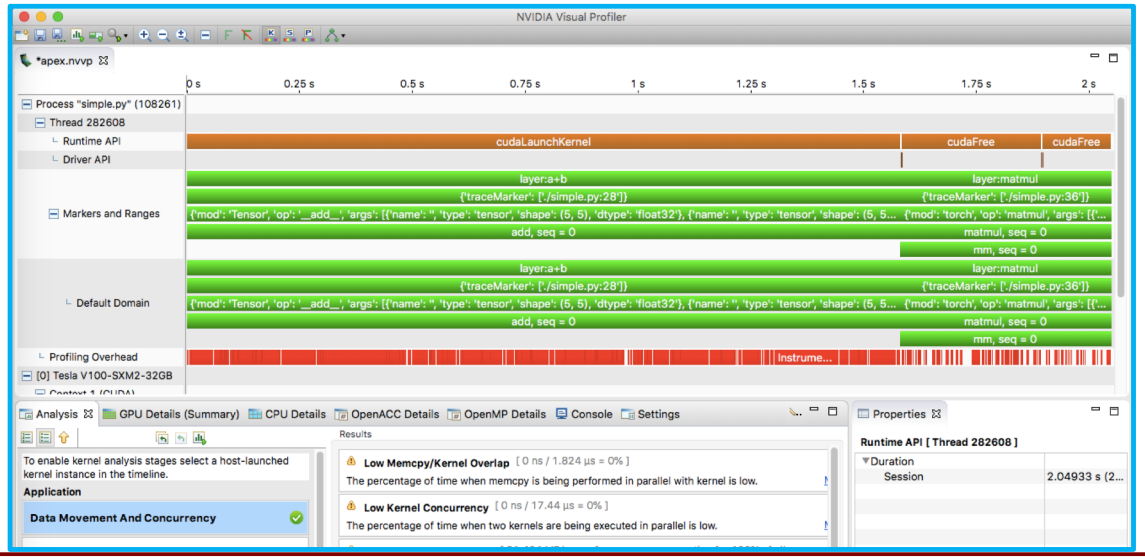
TensorFlow & PyTorch Profiling

- **Direct Profiling:**

  $ nvprof –s –o tf_results.nvvp python ./tensorflow_example

  $ nvprof –s –o torch_results.nvvp python ./pytorch_example

- **NVIDIA APEX: A PyTorch Extension**
  - **Integrates with emit_nvtx context manager**
  - **Only kernels within profiler.start() and profiler.stop() calls are profiled**




```
import torch
import torch.cuda.profiler as profiler
import torch.cuda.nvtx as nvtx
from apex import pyprof

pyprof.nvtx.init()

a = torch.randn(5, 5).cuda()
b = torch.randn(5, 5).cuda()

#Context manager
with torch.autograd.profiler.emit_nvtx():

        #Start profiler
        profiler.start()

        nvtx.range_push("layer:a+b")
        c = a + b
        nvtx.range_pop()

        nvtx.range_push("layer:mul")
        c = torch.mul(a,b)
        nvtx.range_pop()

        nvtx.range_push("layer:matmul")
        c = torch.matmul(a,b)
        nvtx.range_pop()

        #Stop profiler
        profiler.stop()
```

Resources

- **NVIDIA Toolkit Download**
  - https://developer.nvidia.com/cuda-downloads
- **NVProf Documentation**
  - https://docs.nvidia.com/cuda/profiler-users-guide/index.html
- **NVIDIA CUDACasts Episode 19: Guide Performance Analysis with the Visual Profiler**
  - https://devblogs.nvidia.com/cudacasts-episode-19-cuda-6-guided-performance-analysis-visual-profiler/
- **ORNL Slides**
  - https://www.olcf.ornl.gov/wp-content/uploads/2018/12/summit_workshop_Profilers.pdf

- **NVIDIA Nsight Compute**
  - **interactive kernel profiler for CUDA applications**
  - https://developer.nvidia.com/nsight-compute-2019_5