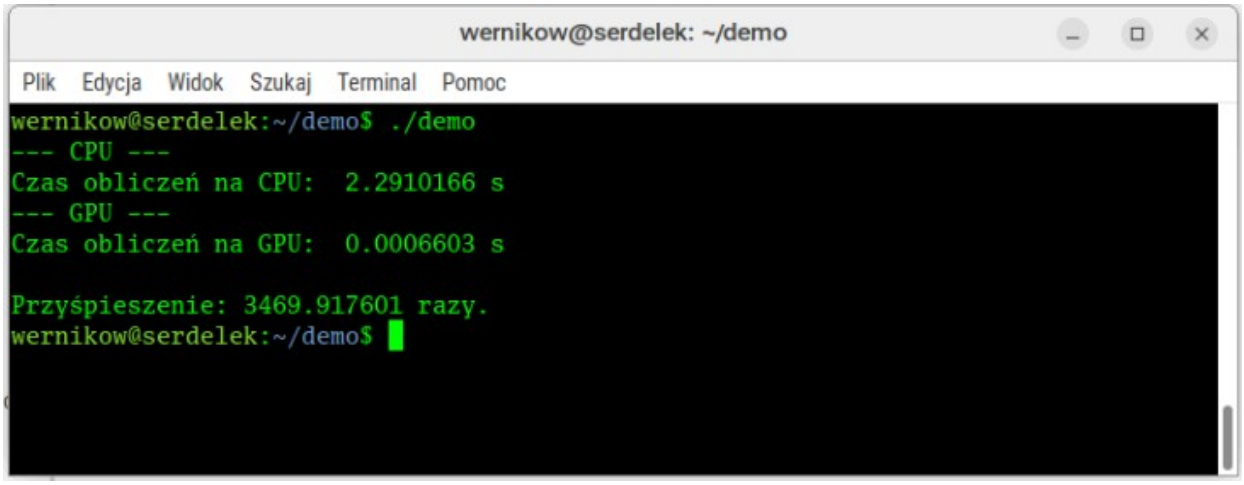
Profiler NVIDIA – szybki start

1. Dla celów niniejszego wprowadzenia użyto prostego kodu z jednym kernelem – treść pliku `demo.cu` znajduje się na końcu tego tekstu.
2. Plik skompilowano do pliku wykonalnego o nazwie `demo` i testowo uruchomiono.

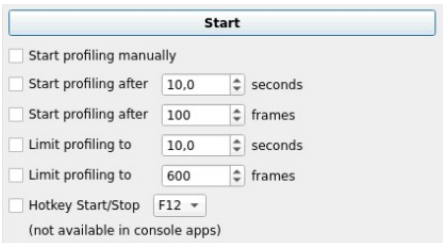


W celu sprofilowania powyższego kodu należy kolejno:

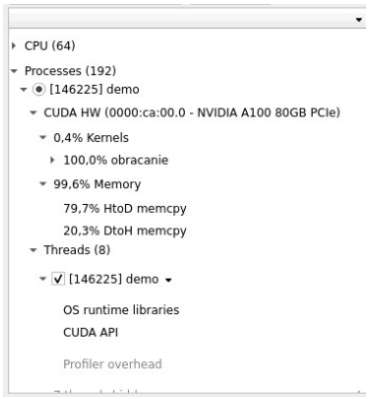
1. uruchomić program *NVIDIA nSight Systems*
2. wykonać sekwencję *File* → *New Project* (tworzony jest projekt o nazwie *Project n*, gdzie *n* jest kolejnym numerem projektu)
3. w oknie po prawej stronie wybieramy *serdelek* jako *target for profiling*
4. pojawiają się ustawienia parametrów profilowania – dwa z nich podlegają obowiązkowemu wypełnieniu:
   1. *command line with arguments* – tutaj wprowadzamy pełną (kanoniczną!) ścieżkę do programu, który chcemy profilować; niedozwolone jest użycie znaku ~ (tylda) dla oznaczenia katalogu domowego;
   2. *working directory* – wypełnia się automatycznie na podstawie tekstu wprowadzonego do poprzedniego pola, ale może wymagać korekty po dokonaniu jakichkolwiek poprawek w pierwszym z pól

5. następnie należy upewnić się, że wśród zbieranych statystyk znajduje się pozycja *Collect CUDA trace*

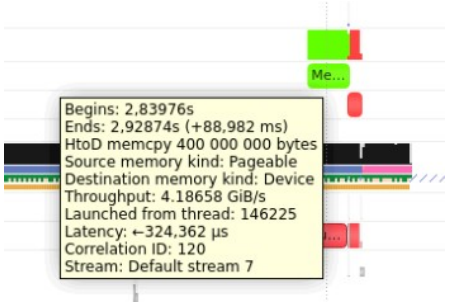
6. w tym momencie można już uruchomić profilowanie klikając w przycisk *Start*



7. Po zakończeniu pracy programu i zignorowaniu ewentualnych ostrzeżeń pojawi się ekran wyników profilowania (może to zająć kilka sekund) – należy w nim rozwinąć gałęzie odpowiedzialne za prezentację aktywności urządzenia CUDA



8. W prawej części wyników ukaże się wykres czasowy, z którego można zgrubnie odczytać proporcje czasów zajętych przez transfery danych z/do urządzenia oraz wykonań kerneli. Dokładne wartości ukażą się po najechaniu kursorem myszy na wybrane odcinki na osi czasu.



Załącznik 1 – kod testowy użyty w przykładzie.

```
#include <stdio.h>
#include <math.h>
#include <assert.h>

#define N 50000000

struct WIERZCHOLEK {
          float x,y;
} Figura[N];

struct timespec stoper_start(){
          struct timespec start;
          clock_gettime(CLOCK_MONOTONIC, &start);
          return start;
}

long stoper_stop(struct timespec start){
    struct timespec koniec;
    clock_gettime(CLOCK_MONOTONIC, &koniec);
    long nanosekundy = (koniec.tv_sec * 1E9 + koniec.tv_nsec) -
                       (start.tv_sec  * 1E9 + start.tv_nsec);
    return nanosekundy;
}

__global__ void obracanie(WIERZCHOLEK *W, float alfa) {
          int i = blockDim.x * blockIdx.x + threadIdx.x;

          if (i < N) {
                    float x = W[i].x * acos(alfa) - W[i].y * asin(alfa);
                    float y = W[i].x * asin(alfa) + W[i].y * acos(alfa);
                    W[i].x = x;
                    W[i].y = y;
          }
}

int main(void) {
          float               alfa = 3.1415;
          struct              timespec stoper;
          long                cpu, gpu;
          int                 ile_cudow;
          WIERZCHOLEK    *d_W;

          puts("--- CPU --- ");
          stoper = stoper_start();
        for(int i = 0; i < N; i++) {
                float x = Figura[i].x * acos(alfa) - Figura[i].y * asin(alfa);
                float y = Figura[i].x * asin(alfa) + Figura[i].y * acos(alfa);
                Figura[i].x = x;
                Figura[i].y = y;
        }
        cpu = stoper_stop(stoper);
        printf("Czas obliczeń na CPU: %10.7f s\n", cpu / 1e9);
          cudaGetDeviceCount(&ile_cudow);
          if(ile_cudow == 0) {
                    perror("Nie ściemniaj – nie masz CUDY");
                    return 1;
          }
          puts("--- GPU ---");
          cudaMalloc(&d_W, sizeof(Figura));
          cudaMemcpy(d_W, Figura, sizeof(Figura), cudaMemcpyHostToDevice);
          int watki_na_blok = 1024;
          int bloki_na_siatke = (N + watki_na_blok - 1) / watki_na_blok;
          stoper = stoper_start();
          obracanie<<<bloki_na_siatke, watki_na_blok>>>(d_W, alfa);
          cudaError_t err = cudaGetLastError();
          assert(err == cudaSuccess);
          cudaDeviceSynchronize();
          gpu = stoper_stop(stoper);
          cudaMemcpy(Figura, d_W, sizeof(Figura), cudaMemcpyDeviceToHost);
          printf("Czas obliczeń na GPU: %10.7f s\n", gpu / 1e9);
          cudaFree(d_W);
          puts("");
          printf("Przyśpieszenie: %f razy.\n", (double)cpu/(double)gpu);
          return 0;
}
```