Witold Alda

Programowanie procesorów graficznych w CUDA. Laboratorium 1.

Kompilujemy program

Kompilacji programów dokonujemy za pomocą kompilatora **nvcc**, który obsługuje przemieszane kody C/C++ i kody CUDY (te ostatnie często, choć nie zawsze zapisywane w plikach z rozszerzeniem .cu).

Proszę obejrzeć parametry wywołania kompilatora, np. za pomocą **nvcc --help**, lub w podobny sposób.

Dwa programy ze standardowej instalacji CUDA

Pakiet instalacyjny dostarcza ponad sto przykładowych programów, które pozwalają na demonstrację prawie wszystkich aspektów programowania w CUDA. Z niektórych będziemy korzystać.
Teraz proszę skompilować i uruchomić dwa takie programy: deviceQuery i bandwidthTest.
Pierwszy z nich podaje podstawowe parametry zainstalowanego GPU. Drugi – szybkość transmisji pomiędzy pamięcią CPU i GPU. Proszę przeczytać wyniki i przeanalizować je – będą przydatne przy kolejnych programach.
Za podstawowe parametry możemy uznać:

- Ilość multiprocesorów
- Taktowanie multiprocesorów oraz pamięci
- Szerokość szyny danych
- Ilość pamieci RAM
- Rodzaj pamięci
- Maksymalne ilości wątków oraz bloków.

Pomiar czasu.

Najprostszym sposobem pomiaru czasu w CUDA jest użycie zestawu funkcji, jak w poniższym przykładzie, wykorzystujące funkcje opisane w **helper_timer.h**:

```
StopwatchInterface *timer=NULL;
sdkCreateTimer(&timer);
sdkResetTimer(&timer);
sdkStartTimer(&timer);

cudaMalloc((void **) &a_d, size);   // alokuj pamięć na GPU
cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
// wykonaj obliczenia na GPU:
kernel <<< n_blocks, block_size >>> (a_d, N);
cudaThreadSynchronize();
```

```
sdkStopTimer(&timer);
float time = sdkGetTimerValue(&timer);
sdkDeleteTimer(&timer);

// prześlij wyniki
cudaMemcpy(a_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
cudaFree(a_d);
printf ("Time for the kernel: %f ms\n", time);
```

Istotne jest przy tym użycie funkcji `cudaThreadSynchronize();` W przeciwnym razie wyniki będą niepoprawne.
Popularny jest również sposób pomiaru oparty na zdarzeniach. Zostanie przedstawiony na następnych zajęciach laboratoryjnych.

Dodawanie wektorów.

Proszę uruchomić program dodawania wektorów na GPU i CPU napisany według poniższego schematu.

```
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>
#define N 10
__global__ void add (int *a,int *b, int *c) {
int tid = blockIdx.x * blockDim.x + threadIdx.x;
if(tid < N) {
c[tid] = a[tid]+b[tid];
}
}
int main(void) {
int a[N],b[N],c[N];
int *dev_a, *dev_b, *dev_c;
cudaMalloc((void**)&dev_a,N * sizeof(int));
cudaMalloc((void**)&dev_b,N * sizeof(int));
cudaMalloc((void**)&dev_c,N * sizeof(int));
for (int i=0;i<N;i++) {
a[i] = i;
b[i] = i*1;
}
cudaMemcpy(dev_a, a , N*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpy(dev_b, b , N*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpy(dev_c, c , N*sizeof(int),cudaMemcpyHostToDevice);
add<<<1,N>>>(dev_a,dev_b,dev_c);
cudaMemcpy(c,dev_c,N*sizeof(int),cudaMemcpyDeviceToHost);
for (int i=0;i<N;i++) {
printf("%d+%d=%d\n",a[i],b[i],c[i]);
}
cudaFree(dev_a);
cudaFree(dev_b);
cudaFree(dev_c);
return 0;
}
```

Proszę uzupełnić kod o pomiar czasu, a następnie uruchomić go próbując różne rozmiary tablicy, różne liczby wątków i bloków. Proszę też zwrócić uwagę na ograniczenia rozmiarów.

W szczególności proszę:
(a) Umożliwić ustalanie rozmiarów tablicy z klawiatury w trakcie wykonania, tak żeby łatwo było zebrać wyniki dla różnych danych;
(b) dodać kod, który wykonuje dodawanie tylko na CPU i podaje czas wykonania – tak żeby można było porównać GPU z CPU.
(c) dodać procedurę, która dla porządku sprawdzi, że GPU I CPU dają takie same wyniki.
(d) Umożliwić zmianę struktury bloków (ile wątków na blok?) i gridów (ile bloków w gridzie?). Proszę ocenić jak wpływa to efektywność program.

Wyniki, po zdroworozsądkowym spojrzeniu na nie, proszę przedstawić na dwóch trzech wykresach, które wraz z krótkim komentarzem będą stanowić sprawozdanie z ćwiczenia.