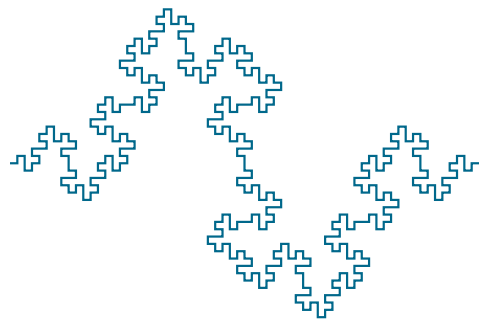
Programowanie równoległe
Wprowadzenie do OpenCL

Rafał Skinderowicz



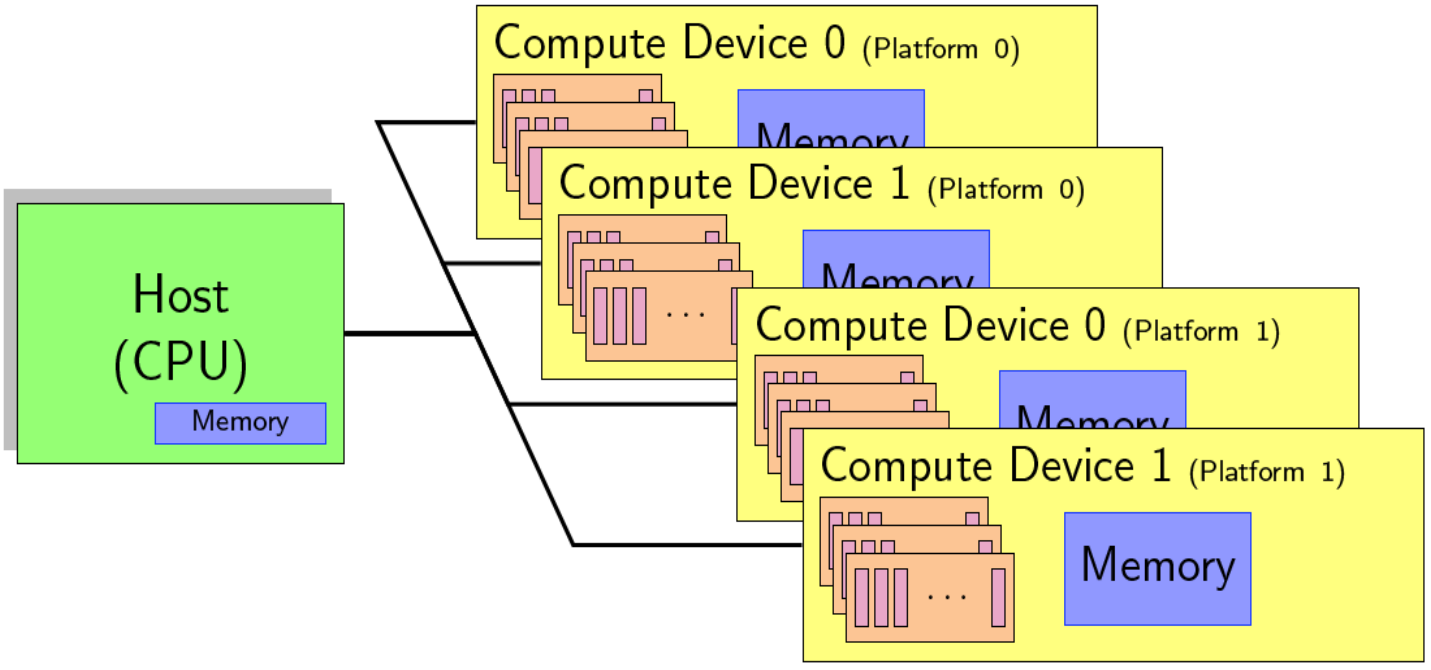
OpenCL – architektura

- OpenCL – Open Computing Language – otwarty standard do programowania *heterogenicznych* platform złożonych ze zbioru CPU, GPU oraz innych urządzeń obliczeniowych
- OpenCL = interfejs programistyczny (API) + biblioteki + system uruchomieniowy (np. sterownik karty graf.)
- Specyfikacja OpenCL definiuje OpenCL za pomocą modeli:
  - platformy
  - wykonania
  - pamięci
  - programowania

OpenCL – platforma

- Platforma w OpenCL [[to]] system główny (host) połączony z jednym lub więcej urządzeniami OpenCL (devices)
- Urządzenie składa się z jednej lub więcej jednostek obliczeniowych (compute units, CU)
- Jednostki obliczeniowe zawierają jeden lub więcej elementów przetwarzania (processing elements, PEs)

OpenCL – platforma

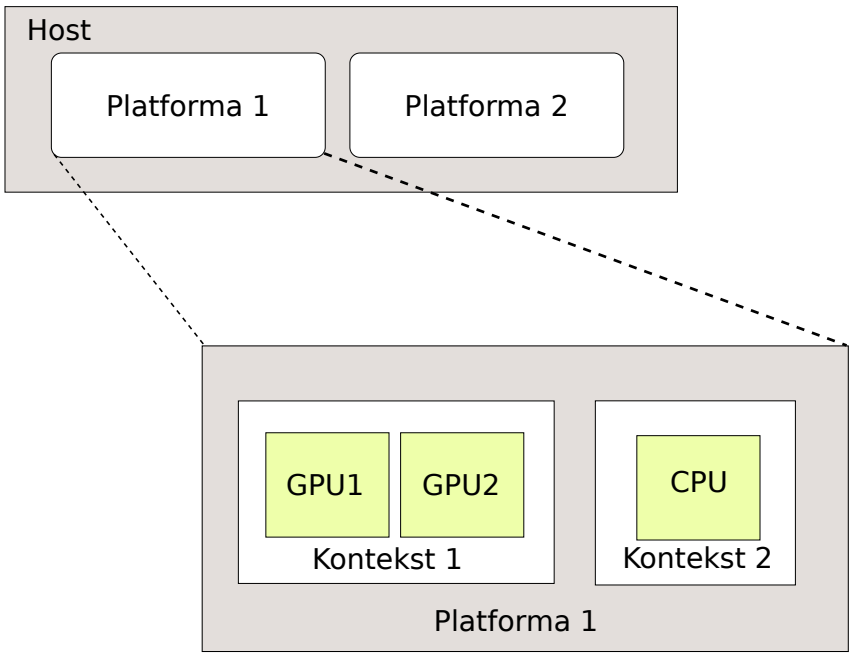


OpenCL – platforma

- Jednostki obliczeniowe (compute units, CU) – zawierają m.in. układy pobierania i dekodowania rozkazów, układy planowania i podziału pracy
- Elementy przetwarzania (processing elements, PE) realizują kolejne instrukcje kernela w modelu SPMD (single program, multiple data)
- Znaczenie CU i PE zależy od konkretnego typu urządzenia, np.:
  - Nvidia: CU = multiprocesor strumieniowy, PE = rdzeń CUDA
  - AMD: CU = jednostka SIMD (w terminologii AMD), PE = SIMD lane
  - CPU: CU = PE = rdzeń procesora

OpenCL – model wykonania

- Program OpenCL uruchamiany jest na hoście, który definiuje kontekst obliczeniowy oraz zarządza wykonaniem na urządzeniach OpenCL wskazanych funkcji – kerneli
- Kontekst to zbiór danych związanych z wykonaniem programu na urządzeniach obliczeniowych
- Kontekst istnieje w ramach plaformy i odnosi się najczęściej do akceleratorów (urządzeń) tego samego typu



Kontekst obejmuje:

- urządzenia OpenCL
- kernele OpenCL
- programy – źródła i binaria z implementacją kerneli
- obiekty pamięciowe

OpenCL – model wykonania

- Host po utworzeniu kontekstu tworzy (jedną lub więcej) powiązaną z nim kolejkę zadań (command queue)
- Kernele (zadania) do wykonania na urządzeniach OpenCL wstawiane są do kolejki
- Kernele mogą być wykonywane w sposób asynchroniczny (out-of-order) i w kolejności niezależnej od kolejności zlecania

OpenCL – model wykonania

- Program OpenCL składa się ze zbioru kerneli
- Kod kerneli pisany jest w zmodyfikowanej wersji języka C (ISO C99)
- Kod kernela kompilowany jest przez system uruchomieniowy OpenCL (np. ster. karty graf.) w trakcie wykonania programu
- Możliwe jest również skomilowanie kodu kernela wcześniej i korzystanie z wersji binarnej

OpenCL – model wykonania – indeksowanie

- Dla każdego kernela definiowana jest dyskretna przestrzeń indeksów (index space)
- Instancja kernela **(wątek, work-item)** wykonywana jest dla każdego punktu przestrzeni indeksów – każdy wątek ma unikalny identyfikator globalny
- Każdy wątek wykonuje współbieżnie kod kernela, jednak przebieg wykonania może różnić się od pozostałych (np. na skutek rozgałęzienia)

OpenCL – model wykonania – indeksowanie

- Każdy wątek należy do jednej z **grup** (work-group), na które podzielona jest przestrzeń indeksów implementowaną w OpenCL przez typ NDRange
- W ramach grupy każdy wątek ma swój lokalny identyfikator – unikalny wew. grupy
- Każda grupa wątków również ma swój unikalny identyfikator

OpenCL – przestrzeń indeksów

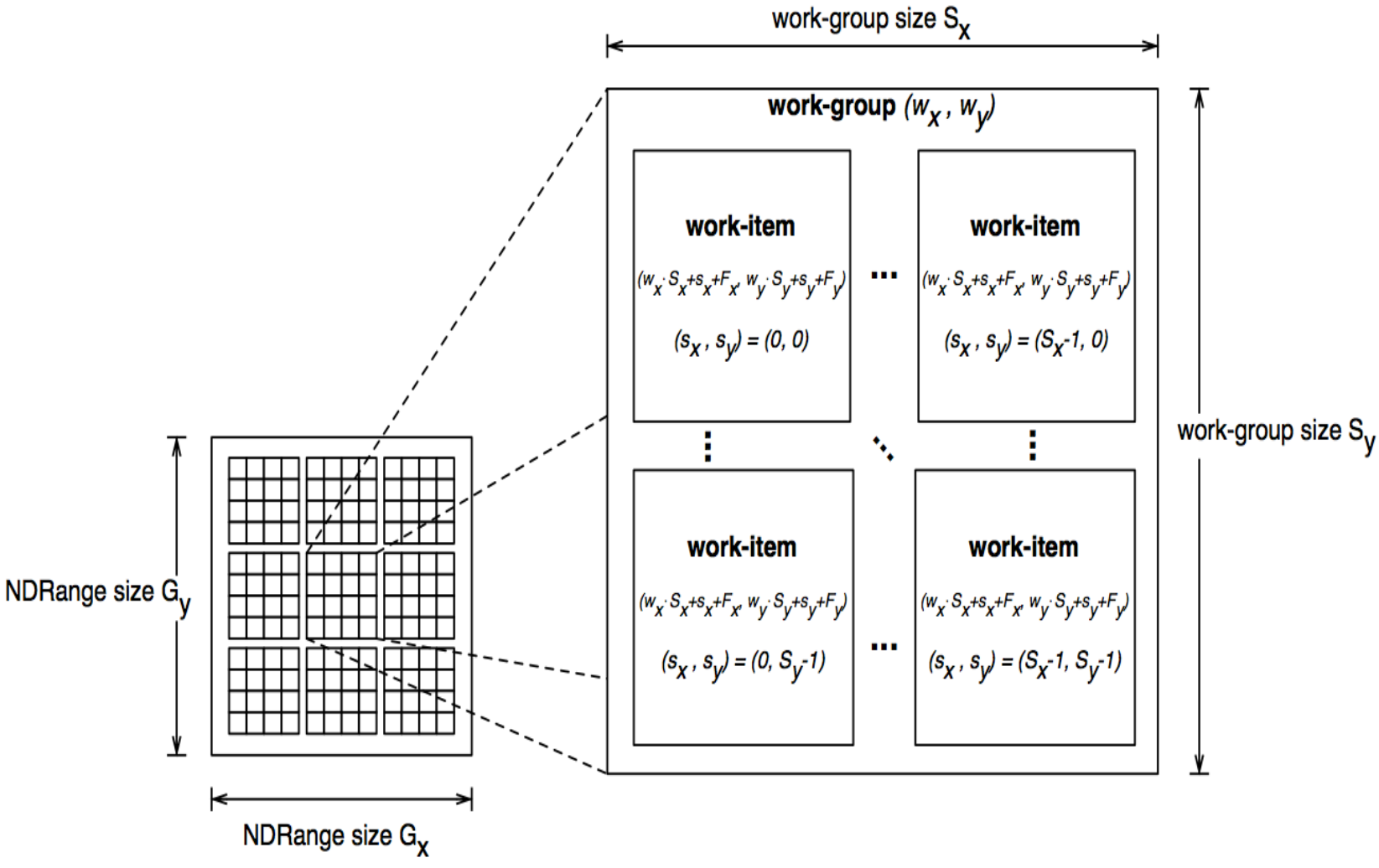
- Przestrzeń indeksów w OpenCL definiowana jest za pomocą typu NDRange, który jest 1-, 2- albo 3-wymiarową tablicą liczb całkowitych określających wartości indeksów w każdym kierunku
- Indeksy globalne i lokalne wątku są N-wymiarowymi krotkami, gdzie $N \in \{1, 2, 3\}$
- Wątek może zostać zidentyfikowany na podstawie:
  - identyfikatora globalnego
  - identyfikatora lokalnego oraz identyfikatora grupy

OpenCL – przestrzeń indeksów

Przykładowo dla 2-wymiarowej przestrzeni indeksów istnieje następująca zależność między identyfikatorem globalnym (g_x, g_y) a lokalnym (s_x, s_y):
$(g_x, g_y) = (w_x \cdot S_x + s_x + F_x, w_y \cdot S_y + s_y + F_y)$,
gdzie:

- (w_x, w_y) to identyfikator grupy
- (S_x, S_y) to rozmiar grupy
- (F_x, F_y) to przesunięcie (offset)

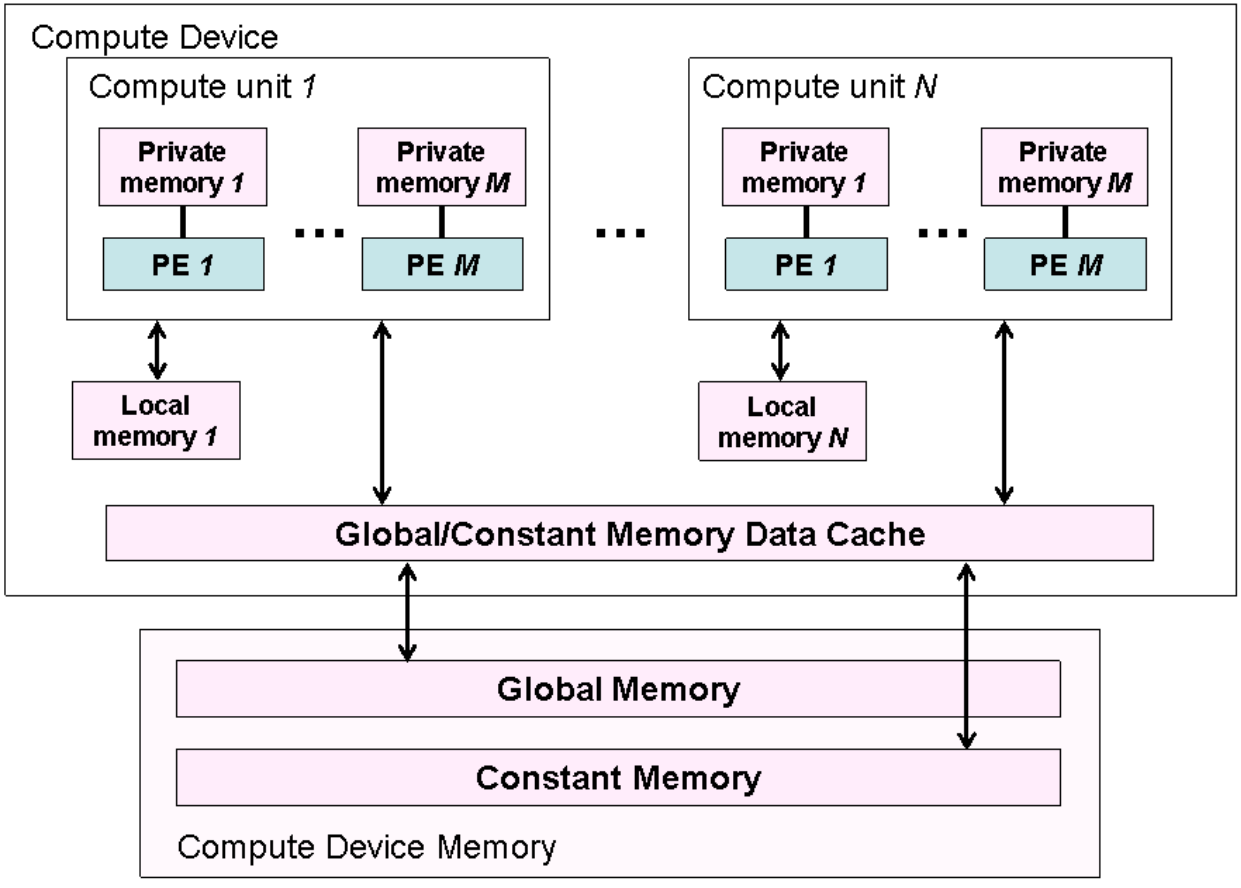
OpenCL – przestrzeń indeksów



OpenCL – model pamięci

- Wątki (work-items) wykonujące kod kernela mają dostęp do czterech typów pamięci:
  - **pamięci globalnej** – dostępna do odczytu i zapisu dla wszystkich wątków we wszystkich grupach
  - **pamięci stałej** – fragment pamięci globalnej tylko do odczytu
  - **pamięci lokalnej** – pamięć dostępna dla wątków *należących do tej samej grupy*
  - **pamięci prywatnej** – każdy wątek ma swoją pamięć prywatną

OpenCL – model pamięci



Rysunek : *Schemat modelu pamięci OpenCL, źródło OpenCL Specification, v1.2*

OpenCL – model pamięci

Dostęp do pamięci				
	Globalna	*Stała*	*Lokalna*	*Prywatna*
Host	Alokacja dynamiczna	Alokacja dynamiczna	Alokacja dynamiczna	Brak
	Odczyt / zapis	Odczyt / zapis	Brak dostępu	Brak dostępu
Kernel	Brak	Alokacja statyczna	Alokacja statyczna	Alokacja statyczna
	Odczyt / zapis	Tylko odczyt	Odczyt / zapis	Odczyt / zapis

- Pamięci hosta i urządzenia są w większości niezależne – konieczny jest transfer danych z jednej do drugiej
- Transfer może odbywać się:
  - za pomocą operacji kopiowania
  - za pomocą odwzorowania (mapowania) z pamięci hosta do pamięci urządzenia

OpenCL – model programownia

- OpenCL wspiera dwa rodzaje równoległości w programach:
- równoległość danych (ang. data parallelism) – sekwencja instrukcji wykonywana jest wielokrotnie, ale dla różnych danych, np. kolejnych punktów obrazu
  - przydział danych dla jednostek przetwarzania
- równoległość zadaniowa (ang. task parallelism) – pojedyncza instancja kernela (zadanie) wykonywana jest dla całej przestrzeni indeksowania

OpenCL – obiekty pamięci

- Złożone dane wejściowe dla kernela oraz wyniki jego wykonania zapisywane są w obiektach pamięci (memory objects)
- OpenCL wyróżnia dwa typy obiektów pamięci:
  - **buforach** – tablica jednowymiarowa
  - **obrazach** – 2- lub 3-wymiarowa tekstura lub obraz
- Bufor jest sekwencją danych typów podstawowych lub zdefiniowanych przez użytkownika
- Obraz to tablica punktów dostępnych dla kernela w postaci 4 elementowych wektorów – wewnętrzny format obrazu może być inny

OpenCL API – inicjalizacja

OpenCL API – inicjalizacja

```
1  #include <CL/cl.h> // użyjemy API z języka C
2  ...
3  int main() {
4  cl_int status; // CL_SUCCESS jeżeli ok, kod błędu wpp.
5  cl_uint num_platforms = 0;
6  // Sprawdź liczbę platform OpenCL
7  status = clGetPlatformIDs(0, NULL, &num_platforms);
8  if (num_platforms == 0 || status != CL_SUCCESS) { return EXIT_FAILURE; }
9  // Pobierz identyfikatory platform
10 std::vector<cl_platform_id> platforms(num_platforms);
11 // Drugi raz clGetPlatformIDs, inne parametry
12 status = clGetPlatformIDs(num_platforms, &platforms.front(), NULL);
```

OpenCL API – inicjalizacja

```
1  // Szukamy platformy z odpowiednim typem urządzenia
2  const cl_device_type kDeviceType = CL_DEVICE_TYPE_GPU;
3  cl_device_id device; // Wystarczy nam pierwsze z brzegu urządzenie
4  cl_platform_id platform;
5  bool found = false;
6  for (cl_uint i = 0; i < num_platforms && !found; ++i) {
7      cl_uint count = 0;
8      status = clGetDeviceIDs(platforms[i], kDeviceType, 1, &device, &
           count);
9      if (count == 1) { // OK, znaleźliśmy
10         platform = platforms[i];
11         found = true;
12     }
13 }
```

OpenCL API – inicjalizacja

```
1  // Tworzymy kontekst obliczeniowy
2  const cl_context_properties prop[] = { // Parametry określające kontekst
3      CL_CONTEXT_PLATFORM, (cl_context_properties)platform,
4      0 // 0 -> znacznik końca listy parametrów
5  };
6  cl_context context = clCreateContextFromType(prop, kDeviceType, NULL,
7                                               NULL, &status);
8  if (status != CL_SUCCESS) {
9      cout << "Problem z tworzeniem kontekstu obliczeniowego. Kod błędu: "
10          << status << endl;
11 }
12 // Tworzymy kolejkę poleceń OpenCL
13 cl_command_queue cmd_queue = clCreateCommandQueue(context,
14                                                    device,
15                                                    0, &status);
```

OpenCL API – program OpenCL

```
1  // Nasz program w postaci łańcucha
2  const char *src = ""
3      "__kernel void example(__global const float* A,"
4      " int k,"
5      " __global float* C) {"
6      " int id = get_global_id(0);"
7      " C[id] = A[id] * k;"
8      "}";
9  // Tworzymy obiekt programu OpenCL
10 cl_program program = clCreateProgramWithSource(context, 1,
11                                                 (const char**)&src,
12                                                 NULL, NULL);
13 status = clBuildProgram(program, 1, devices, NULL, NULL, NULL);
14 if (status != CL_SUCCESS) {
15     char log[1024] = {};
16     clGetProgramBuildInfo(program, devices[0], CL_PROGRAM_BUILD_LOG,
17                           1024, log, NULL);
18     cout << "Build log:\n" << log << endl;
19     return EXIT_FAILURE;
20 }
```

OpenCL API – kernel

```
1  cl_kernel kernel = clCreateKernel(program, "example", NULL);
2  // Dane wejściowe dla obliczeń
3  cl_float scores[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
4  const int n = sizeof(scores) / sizeof(scores[0]);
5  // Alokacja buforów na dane wejściowe i wyniki obliczeń kernela
6  cl_mem in_mem = clCreateBuffer(context, CL_MEM_READ_ONLY |
       CL_MEM_COPY_HOST_PTR,
7                                 sizeof(cl_float4) * n, scores, NULL);
8  cl_mem out_mem = clCreateBuffer(context, CL_MEM_READ_WRITE,
9                                 sizeof(cl_float) * n, NULL, NULL);
10 cl_int k = 20; // Parametr typu podstawowego
11 // Przekaż parametry dla kernela
12 clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *) &in_mem);
13 clSetKernelArg(kernel, 1, sizeof(cl_int), (void *) &k);
14 clSetKernelArg(kernel, 2, sizeof(cl_mem), (void *) &out_mem);
```

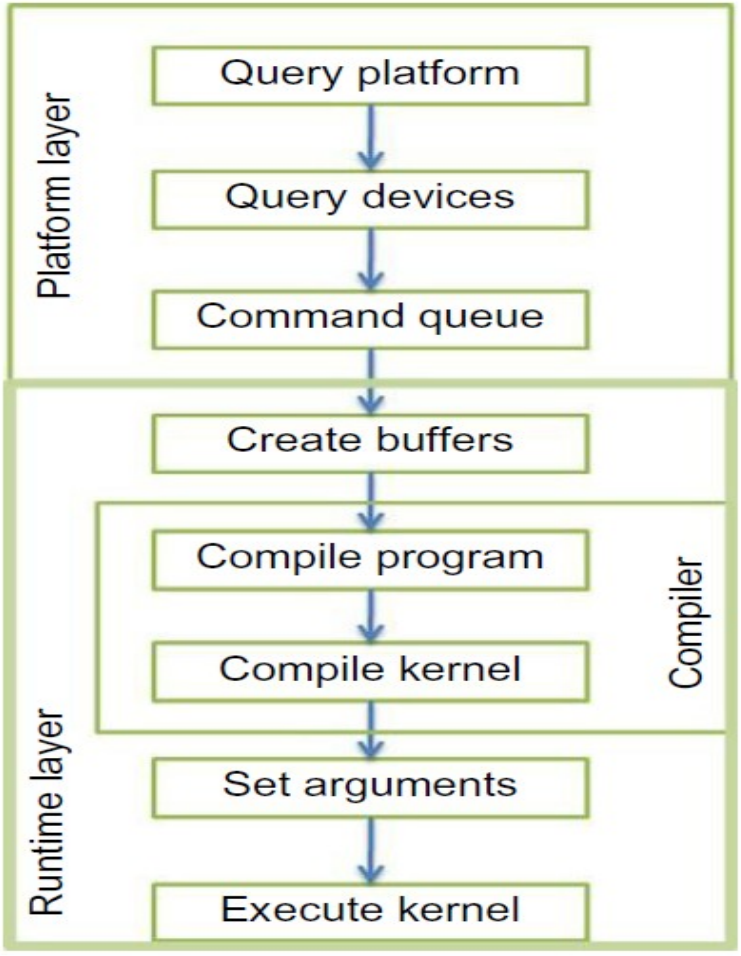
OpenCL API – wykonanie kernela

```
1  const int dimensions = 1;
2  // Definiujemy przestrzeń indeksowania
3  size_t global_work_size[dimensions] = { n };
4  cl_event event;
5  status = clEnqueueNDRangeKernel(cmd_queue, kernel, dimensions, NULL,
6                                  global_work_size, NULL, 0, NULL, &event);
7  clWaitForEvents(1, &event); // Czekaj na wykonanie kernela
8  // Kopiujemy wynik z pamięci urządzenia do pamięci hosta
9  cl_float result[n] = {};
10 status = clEnqueueReadBuffer(cmd_queue, out_mem, CL_TRUE,
11                               0, n * sizeof(cl_float), result, 0, NULL, NULL
                                      );
12 // Pokaż wynik
13 cout << "Result: ";
14 for (int i = 0; i < n; ++i) {
15     cout << result[i] << " ";
16 }
17 cout << endl;
```

OpenCL API – sprzątanie

```
1 clFinish(cmd_queue); // Czekamy na zakończenie zadań
2 // Zwalniamy pamięć
3 clReleaseMemObject(in_mem);
4 clReleaseMemObject(out_mem);
5 clReleaseKernel(kernel);
6 clReleaseProgram(program);
7 clReleaseCommandQueue(cmd_queue);
8 clReleaseContext(context);
```

OpenCL API – podsumowanie sekwencji operacji



Funkcje dotyczące przestrzeni indeksowania

- `uint get_work_dim()` – zwraca liczbę wymiarów przestrzeni indeksowania (1,2 lub 3)
- `size_t get_global_size(uint dim)` – zwraca globalną liczbę wątków dla wymiaru `dim`
- `size_t get_global_id(uint dim)` – zwraca globalny identyfikator wątku dla wymiaru `dim`
- `size_t get_local_size(uint dim)` – zwraca liczbę wątków dla wymiaru `dim` w grupie, do której należy wątek
- `size_t get_local_id(uint dim)` – zwraca identyfikator wątku dla wymiaru `dim` wew. grupy, do której należy wątek
- `size_t get_num_groups(uint dim)` – zwraca liczbę grup dla podanego wymiaru
- `size_t get_group_id(uint dim)` – zwraca identyfikator grupy dal podanego wymiaru
- `size_t get_global_offset(uint dim)` – zwraca przesunięcie (offset) dla podanego wymiaru; domyślnie 0

Funkcje dotyczące przestrzeni indeksowania

```
// Przestrzeń indeksowania definiujemy przy zlecaniu zadania
const size_t dim = 2;
size_t offset[] = { 0, 0 };
size_t global_threads[] = { 100, 50 }; // 100x50 wątków
size_t local_threads[] = { 16, 16 }; // Grupy 16x16
...
status = clEnqueueNDRangeKernel(cmd_queue, kernel,
                                dim,
                                offset, // może być NULL dla {0, 0}
                                global_threads,
                                local_threads,// dla NULL automatycznie
                                0, NULL, &event);
```

```
// Każdy wątek wykonujący kernel otrzymuje indeks w przestrzeni
// indeksowania
size_t x = get_global_id(0);
size_t y = get_global_id(1);
// Albo
size_t x = get_group_id(0) * get_local_size(0) + get_local_id(0) +
           get_global_offset(0);
size_t y = get_group_id(1) * get_local_size(1) + get_local_id(1) +
           get_global_offset(1);
```

OpenCL a CUDA

CUDA	**OpenCL**
Krata (grid)	Przestrzeń indeksowania (index space)
Blok (block)	Grupa zadań (work-group)
Wątek (thread)	Zadanie (work-item)