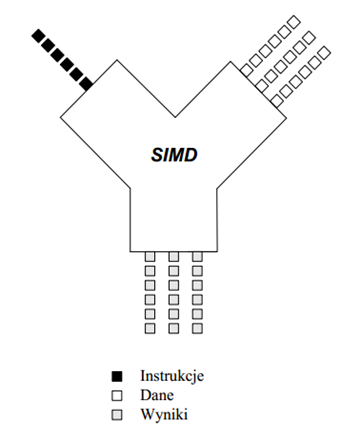
**1. Opisać cechy rolę i znaczenie w masywnych obliczeniach równoległych: Rejestrów wektorowych w architekturze procesora i obliczeń równoległych**

**Rejestry wektorowe** - przechowujące dane do jednoczesnego przetwarzania wielu danych przez instrukcje typu SIMD.

**Systemy SIMD** (ang Single Instruction Multiple Data) to komputery wektorowe, realizujące rozkazy przetwarzające zbiory danych (wektor). W ten sposób zamiast realizować sekwencyjne operacje na poszczególnych elementach zbioru danych, komputer SIMD w jednym rozkazie wykonuje operację na zbiorze danych o określonej liczebności.



Rys. 1 Logiczna budowa systemów SIMD

Główne korzyści wynikające z architektury SIMD osiągają aplikacje, w których te same operacje arytmetyczne lub logiczne, są wykonywane na dużych zbiorach danych.

**Architektura SIMD na dwa sposoby przyspiesza ten proces:**

* Po pierwsze, odczyt i zapis danych do pamięci odbywa się blokowo i określona liczba danych zostaje odczytana/zapisana w jednej operacji
* Po drugie, określona operacja arytmetyczna lub logiczna jest jednocześnie wykonywana na całym bloku danych, podczas wykonania jednej instrukcji

**Obliczenia równoległe** są formą obliczeń numerycznych, w których wiele operacji przeprowadza się jednocześnie przez równolegle pracujące jednostki wykonawcze. Możliwość przeprowadzania obliczeń równoległych wynika z zasady, że problemy numeryczne wymagające wielkiej liczby operacji mogą być z zasady podzielone na problemy o mniejszej złożoności, które z kolei można rozwiązywać równocześnie.

**Najważniejszymi zaletami obliczeń równoległych są:**

- przyśpieszenie obliczeń

- zwiększenie niezawodności działania lub dokładności

- stworzenie możliwości rozwiązania zadań zbyt dużych dla maszyn sekwencyjnych

- umożliwienie wykorzystania pełnej mocy (tzn. łącznej mocy wszystkich rdzeni) procesorów wielordzeniowych.

**Przykłady zastosowanie obliczeń równoległych:**

- obliczanie prognozy pogody

- modelowanie zmian klimatu w skali globalnej np. wpływ efektu cieplarnianego

- badanie własności związków chemicznych, poczynając od modeli na poziomie atomów i cząsteczek

- projektowanie złożonych układów scalonych bardzo dużej skali integracji

- badania w medycynie i genetyce np. poznawanie ludzkiego genomu

- zadania biologii molekularnej, w tym modelowanie zmian zachodzących w komórkach pod wpływem pewnych mutacji DNA, infekcji wirusowych, promieniowania, starzenia oraz modelowanie zmian struktury białek, do których dochodzi w wyniku różnego typu stresu, np. termicznego lub chemicznego

- obliczenia w astronomii i kosmologii

- modele przepływów turbulentnych

- modelowanie wybuchów jądrowych i ich skutków

- badania aerodynamiki oraz wytrzymałości samochodów, samolotów i rakiet

- obliczenia konstrukcyjne dużych budynków, mostów i zbiorników

- modelowanie oceanu np. przemieszczanie się prądów i zanieczyszczeń, fal tsunami

- badania w fizyce

- szybkie przeszukiwanie bardzo dużych baz danych z realizacją wielu zaleceń naraz np. w wyszukiwarkach internetowych

**2. Opisać cechy, rolę i znaczenie w masowych obliczeniach równoległych: hierarchii i organizacji pamięci, w szczególności pamięci podręcznej**

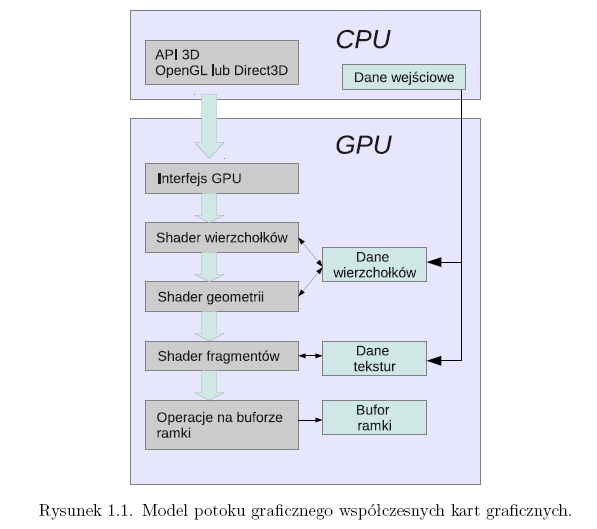
**3.Opisać cechy, znaczenie i rolę w masywnych obliczeniach równoległych: Optymalizacji programów uwzględniającej różne aspekty architektur.**

**4. Opisać cechy, rolę i znaczenie w masowych obliczeniach równoległych: karty graficzne ogólne przeznaczenie GPGPU i ich architektury**

**GPGPU** – obliczenia ogólnego przeznaczenia na układach GPU– technika, dzięki której GPU, zwykle zajmujący się tylko obliczeniami związanymi z grafiką komputerową, umożliwia wykonywanie obliczeń ogólnego przeznaczenia, tak jak CPU. Dzięki temu wiele obliczeń, głównie obliczenia równoległe, można przeprowadzić znacznie szybciej.

**Architektura urządzeń GPU**

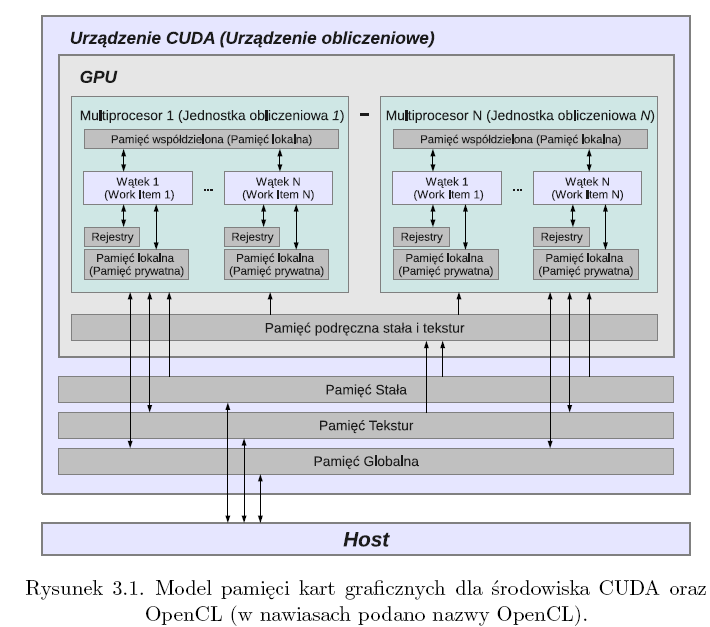
Współczesne procesory graficzne umożliwiają generowanie, w czasie rzeczywistym, realistycznej grafiki dzięki wprowadzeniu programowalnych jednostek realizujących potok graficzny w miejsce ich statycznych odpowiedników. Uproszczony model takiego potoku został pokazany na rysunku 1.1.



Do komunikacji *hosta* z kartą graficzną służy specjalnie zaprojektowane API (ang. *Application Programming Interface*), umożliwiające realizację typowych zadań związanych z generowaniem grafiki. Sam potok graficzny składa się z kilku etapów. Dane wierzchołków, z których składają się podstawowe prymitywy graficzne, po skopiowaniu z pamięci *hosta* są najpierwprzetwarzane przez specjalny program w obrębie tzw. shadera wierzchołków. Jego celem jest obliczenie odpowiednio przetransformowanej pozycji każdego z wierzchołków w przestrzeni trójwymiarowej oraz ich oświetlenie i pokolorowanie. Tak przekształcone wierzchołki trafiają następnie do programu przetwarzającego geometrię, którego celem jest konstrukcja większych prymitywów. Po rasteryzacji powstałych w ten sposób prymitywów, do pracy rusza kolejna jednostka cieniująca, zwana shaderem fragmentów, której celem jest obliczenie koloru każdego punktu danego prymitywu. W tym shaderze możliwe jest wykorzystanie innej porcji danych pochodzących z pamięci *hosta*, tzw. tekstur, stanowiących dwu- lub trójwymiarowe obrazy. Przygotowany w ten sposób obraz zapisywany jest w pamięci bufora ramki i zazwyczaj wyświetlany na ekranie.

**Typy pamięci**

**T**ypy pamięcimożna podzielić na pamięć *hosta* (*host memory*) znajdującą sięfizycznie w przestrzeni CPU oraz pamięć karty graficznej (*device memory*) rezydującą w przestrzeni GPU. Transfer pomiędzy tymi dwoma typami pamięciami jest możliwy tylko przy pomocy dedykowanych funkcji API danego środowiska. Pamięć urządzenia jest jednak zdecydowanie bardziej złożona niż w przypadku *hosta* i ogólnie składa się z pamięci globalnej, lokalnej, stałej, współdzielonej, pamięci tekstur oraz rejestrów. Rysunek 3.1 obrazuje podział pamięci urządzenia obliczeniowego oraz możliwe przepływy danych pomiędzy poszczególnymi typami pamięci.



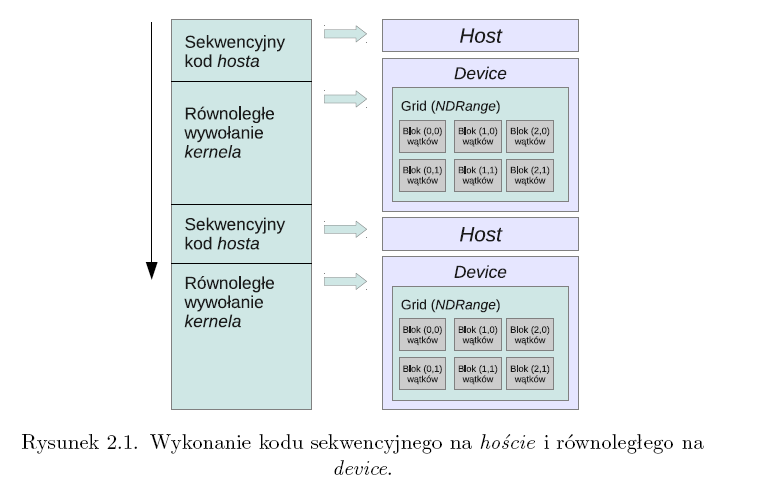
Takie zróżnicowanie jest wynikiem kompromisu pomiędzy szybkością transferu/czasem dostępu do pamięci a wielkością danego typu pamięci. Część typów pamięci (współdzielona, rejestry, podręczna) zostały fizycznie umieszczone wewnątrz procesora GPU zapewniając maksymalną wydajność, natomiast pozostałe, tj. pamięć globalna, tekstur, stała oraz lokalna zostały umieszczone na zewnątrz kości procesora ale są najbardziej pojemne i jednocześnie najwolniejsze. W niektórych architekturach sprzętowych te zewnętrzne pamięci mogą być buforowane w pamięci podręcznej znajdującej się wewnątrz procesora GPU.

**5. Opisać cechy, rolę i znaczenie w masowych obliczeniach równoległych: środowiska i języka programowania kart GPGPU**

**Język CUDA C** jest oparty na standardzie języka C rozszerzając jednocześnie ten standard o dodatkowe typy danych, wbudowane zmienne oraz funkcje, które mogą być wykonywane równolegle na procesorze zgodnym z architekturą CUDA. Wprowadza również nową składnię definiowania meta-parametrów funkcji rdzenia. Niniejszy rozdział przedstawia jedynie te dodatkowe elementy rozszerzające własności standardowego języka C.

**Język OpenCL C** służy do tworzenia programów wykonujących wysoce zrównoleglone funkcje rdzeni. Funkcje te mogą zostać wykonane na wielu heterogenicznych urządzeniach, takich jak CPU, GPU czy innych dedykowanych akceleratorach. Sam program OpenCL można opisać przez podobieństwo do biblioteki łączonej dynamicznie, a funkcje kerneli do funkcji eksportowanych przez taką bibliotekę. Jednakże, w przypadku biblioteki łączonej dynamicznie, funkcje przez nią eksportowane mogą być wywoływane przez *hosta* bezpośrednio, natomiast w przypadku OpenCL funkcje kernela są kolejkowane w kolejce poleceń na urządzeniu obliczeniowym i wykonywane asynchronicznie razem z kodem *hosta*.

**Typowe wykonanie programu CUDA/OpenCL** jest przedstawione na rysunku 2.1.



Całym procesem wykonywania programu steruje procesor *hosta* (CPU). Kiedy zostaje uruchomiona funkcja kernela, wykonywanie przenosi się do urządzenia GPU. W sytuacji, gdy wywołanie funkcji rdzenia odbywa się asynchronicznie, sterowanie powraca od razu do wątku *hosta*. W innym przypadku, wątek *hosta* jest blokowany i czeka na powrót sterowania do czasu zakończenia obliczeń wykonywanych na *urządzeniu*. Po uruchomieniu kernela, na karcie grafiki, tworzony jest zbiór wątków nazywanych *Siatką* (ang. *Grid*) w CUDA lub *NDRange* w OpenCL wykonywanych równolegle

porcjami, w zależności od możliwości danego urządzenia obliczeniowego. Po wykonaniu funkcji kernela przez każdy z wątków *Grid/NDRange* jest usuwany.

