**CUDA**  
  
CUDA jest model równoległy, platforma obliczeniowa oraz interfejs programowania aplikacji (API), stworzony przez firmę NVIDIA. Pozwala programistom używać procesora graficznego (GPU) do przetwarzania ogólnego przeznaczenia. Podejściem znanym jako GPGPU. Platforma CUDA to warstwa oprogramowania, który daje bezpośrednio dostęp do wirtualnego zestawu instrukcji GPU i równoległych elementów obliczeniowych, za wykonanie obliczeń kernela.

Platforma CUDA jest przeznaczona do pracy z C, C ++ i Fortran. Ułatwia w programowaniu równoległym do wykorzystania zasobów GPU, w przeciwieństwie do poprzednich rozwiązań, takich jak API Direct3D i OpenGL, co wymagało zaawansowanych umiejętności w programowaniu graficznym. Ponadto, CUDA obsługuje ramy programowe, takie jak OpenACC i OpenCL.CUDA nazwa Compute Unified Device Architecture.   
  
CUDA ma kilka zalet w stosunku do tradycyjnych obliczeń ogólnego przeznaczenia na GPU (GPGPU) przy użyciu interfejsów API grafiki:

- kod można odczytać z dowolnych adresów w pamięci  
- Ujednolicona pamięć wirtualna (CUDA 4.0 i wyżej)  
- Pamięć Unified (CUDA 6.0 i wyżej)  
- Wspólna pamięć - współdzielony obszar pamięci, które może być dzielony pomiędzy wątkami.  
- Szybsze pobrania i odczyty danych z powrotem z miejsca, w którym został zapisany do GPU  
- Pełne wsparcie dla operacji całkowitych i bitowych  
  
Cuda a OpenCL - Open Computing Language – [framework](https://pl.wikipedia.org/wiki/Framework) wspomagający pisanie aplikacji działających na [heterogenicznych platformach](https://pl.wikipedia.org/wiki/Heterogeniczno%C5%9B%C4%87) składających się z różnego rodzaju [jednostek obliczeniowych](https://pl.wikipedia.org/wiki/Procesor). Główną zaletą jest to, że można użyć jednego otwartego standardu zamiast zamkniętych wspierających sprzęt tylko jednego producenta (np. [CUDA](https://pl.wikipedia.org/wiki/CUDA) tylko dla kart graficznych produkowanych przez [Nvidia](https://pl.wikipedia.org/wiki/Nvidia" \o "Nvidia)).  
  
**Branch Predication – Predykcja**

Jest to obwód cyfrowy, który próbuje odgadnąć jaki sposób będzie dział, (np if-then-else struktury) pójdzie przed tym, gdy do niego dotarł. Celem predyktora gałęzi jest poprawa przepływu w potoku instrukcji. Jest to kluczowa rola w osiągnięciu wysokiej efektywności pracy w wielu nowoczesnych potokowych architektur mikroprocesorów, takich jak x86, strategia w architekturze komputerowego projektowania łagodzenia kosztów związanych zazwyczaj z instrukcjami warunkowych, zwłaszcza odgałęzień do krótkich odcinków kodu. Czyni to poprzez umożliwienie każdej instrukcji warunkowo albo wykonać operację lub nic.   
  
Większość programów komputerowych zawiera kod warunkowy, który zostanie wykonany tylko w określonych warunkach, w zależności od czynników, które nie mogą być ustalone wcześniej, na przykład w zależności od danych wprowadzonych przez użytkownika. Jak większość procesorów wystarczy wykonać następną instrukcję w sekwencji, tradycyjny rozwiązaniem jest wstawienie instrukcji branżowych (predykcji), które pozwalają program warunkowo skoczyć do innej sekcji kodu, zmieniając w ten sposób kolejny krok w sekwencji. Poprawa wydajności kodu poprzez implementację obsługi potoku, metodę, która jest spowalniana przez gałęzie. Nie gwarantuje szybsze wykonanie kodu w ogóle, czasem są za krótkie bloki kodu.  
(Warunek) to zrobić  
(Nie warunek) to zrobić

- Główny powód to zwiększenie efektywności realizacji potoków i uniknięcia problemów z pamięcią podręczną.   
- Funkcje, które są tradycyjnie obliczane za pomocą prostych operacji arytmetycznych i bitowych mogą być szybciej obliczane korzystając z instrukcji warunkowych.  
- Opiera się instrukcje o predykaty więc można mieszać ze sobą kod bezwarunkowy i warunkowy, umożliwiając lepsze harmonogramowanie obsługi i lepsze osiągi.  
- Eliminacja zbędnych instrukcji może sprawić, że wykonanie niezbędnych instrukcji, takich jak te, które tworzą pętle, zmniejszając obciążenie mechanizmów przewidywania rozgałęzień.

**VLIW**

Każda instrukcja koduje tylko jedną operację. Dla większości superskalarnych wzorów, szerokość instrukcja ma 32 bity lub mniej. Natomiast jedna instrukcja VLIW koduje wielokrotne operacje; W szczególności, jedna instrukcja koduje co najmniej jedną operację dla każdej jednostki wykonania urządzenia. Na przykład, jeżeli urządzenie ma pięć VLIW jednostek wykonawczych, to instrukcja VLIW do tego urządzenia ma pięć pól operacyjnych, gdyż każde pole określa jaka operacja powinna być wykonana. Aby zmieścić te pola, instrukcje VLIW są zwykle moją co najmniej 64 bitów szerokości, a na niektórych architekturach są znacznie szersze.   
  
VLIW odnosi się do architektury procesorów przeznaczonych do wykorzystania równoległości na poziomie instrukcji (ILP). W przypadku tradycyjnych jednostek przetwarzania (CPU, procesor) głównie umożliwiają to programy do określenia instrukcji wykonania w sekwencji, a tylko procesor VLIW umożliwia wyraźne określanie instrukcji do wykonania w tym samym czasie, jednocześnie, równocześnie. Konstrukcja ta umożliwia lepsze osiągi bez komplikacji.

**CMS**

Code Morphing Softwarejest to technologia stosowana przez mikroprocesory do wykonywania instrukcji x86., CMS czyta instrukcje i generuje instrukcje zastrzeżone dla procesora VLIW. Tłumaczenie CMS jest znacznie droższe niż Shade, ale daje znacznie wyższą jakość kodu. CMS zawiera również tłumacza i stymuluje zarówno w trybie użytkownika i tryb pracy systemu.  
  
Jest to podejście stosowane w oprogramowaniu do ochrony aplikacji od inżynierii odwrotnej, analizy, modyfikacji i łamania. Technologia ta chroni kodu pośrednie szczeble, jak z Java i C#, a nie binarnego kodu wynikowego. Jest wielopoziomowa technologia zawierająca setki unikalnych wzorów transformacyjnych kodu. Ponadto technologia ta zamienia kilka warstw pośrednich poleceń do poleceń maszyn wirtualnych (jak p-code). Kod morfingu nie chroni przed śledzeniem wykonawczego, który może objawić logikę wykonanie dowolnego kodu chronionego. Code Morphing jest często używany w zabezpieczeniu przed kopiowaniem lub innych kontroli.

**RAID**

**RAID** (ang. Redundant Array of Independent Disks, Nadmiarowa macierz niezależnych dysków) – polega na współpracy dwóch lub więcej dysków twardych w taki sposób, aby zapewnić dodatkowe możliwości, nieosiągalne przy użyciu jednego dysku jak i kilku dysków podłączonych jako oddzielne.

RAID używa się w następujących celach:  
- zwiększenie niezawodności (odporność na awarie),  
- zwiększenie wydajności transmisji danych,  
- powiększenie przestrzeni dostępnej jako jedna całość.

Podczas projektowania macierzy RAID uwzględniane są różnorodne zastosowania [pamięci masowej](https://pl.wikipedia.org/wiki/Pami%C4%99%C4%87_masowa) w [systemach komputerowych](https://pl.wikipedia.org/wiki/System_komputerowy). Przeznaczenie macierzy implikuje wybór odpowiednich technologii w zakresie dysków, kontrolerów, [pamięci podręcznej](https://pl.wikipedia.org/wiki/Pami%C4%99%C4%87_podr%C4%99czna), sposobu przesyłania danych oraz poziomu niezawodności (odpowiedniej nadmiarowości/redundancji podzespołów i połączeń). W macierzach RAID stosuje się wszystkie produkowane obecnie rodzaje dysków twardych. Dominują jednak rozwiązania oparte na serwerowych. Rośnie udział dysków [SSD](https://pl.wikipedia.org/wiki/Solid_State_Drive) w rozwiązaniach wymagających krótkiego czasu dostępu do rozproszonych danych.  
RAID 0 – kolejne bloki danych zapisywane są na dwóch lub więcej oddzielnych dyskach. Zwiększanie szybkości odczytu/zapisu nie gwarantuje żadnej ochrony  
RAID 1 – inaczej mirroring, bloki zapisywane są jednocześnie na dwóch lub więcej oddzielnych dyskach, tworząc efekt odbicia lustrzanego, jest redundancja danych, zwiększona ochrona, szybkość zależy od sterowników   
RAID 5 – bloki danych zapisywane są tylko raz na kolejnych dyskach, każdy dysk zawiera parity czyli dane pozwalające się odzyskać przy awarii jednego z dysków, czyli XOR pozostałych.

RAID 6 – podobny do RAID 5, z tym, że są tu dwa bloki parity, więc nawet odzyskanie danych po awarii dwóch dysków  
RAID 0+1 – połączenie RAID 0 oraz 1, najbardziej kosztowny dyskowo, szybkość RAID 0 niezawodność RAID 1  
RAID 10 – podobnie jak RAID 0+1 lecz tutaj jest stripe mirrorów, lepiej sobie radzi z obudową macierzy