Wprowadzenie do programowania GPU

Marcin Copik

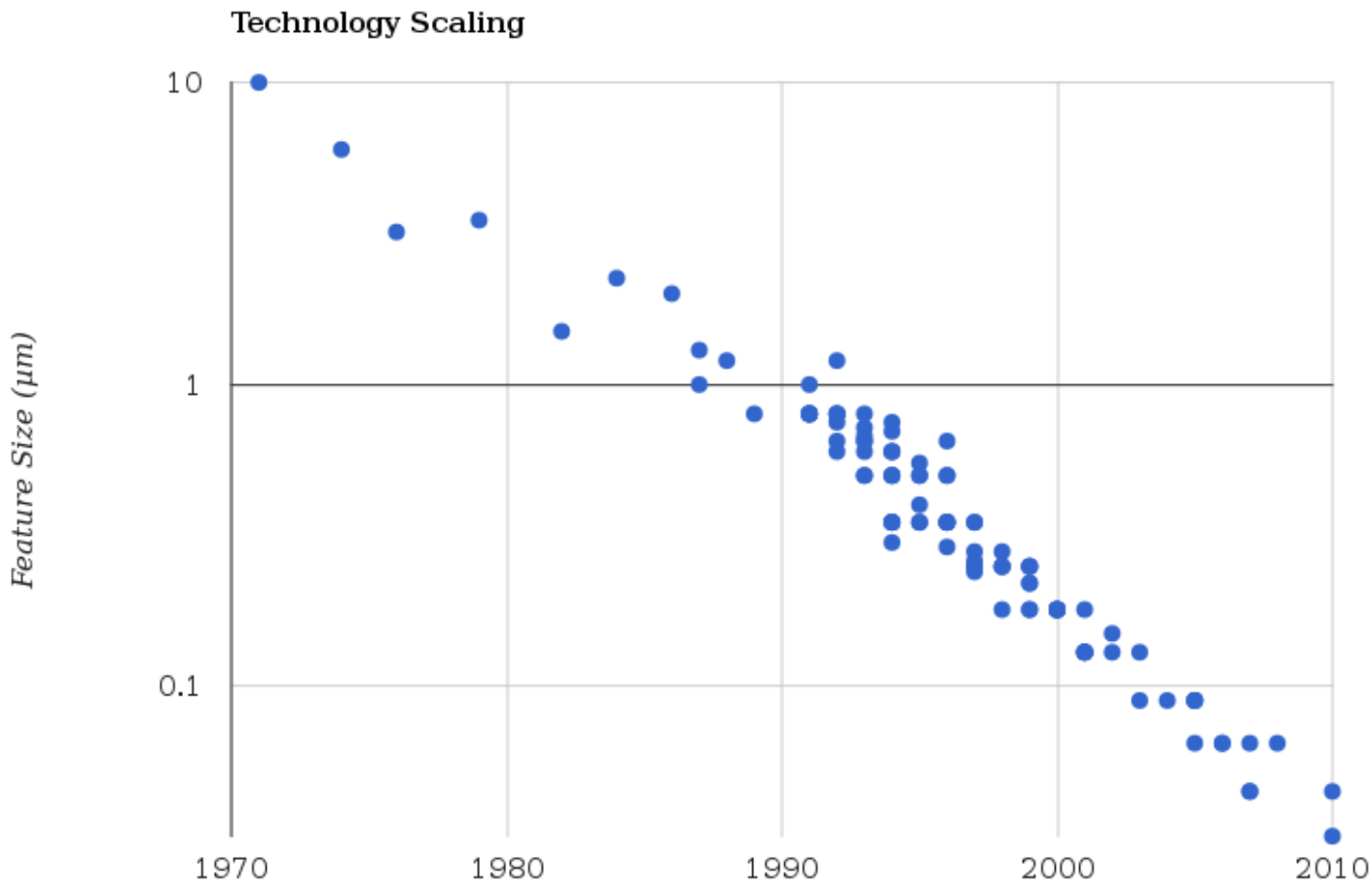
22 stycznia 2017

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

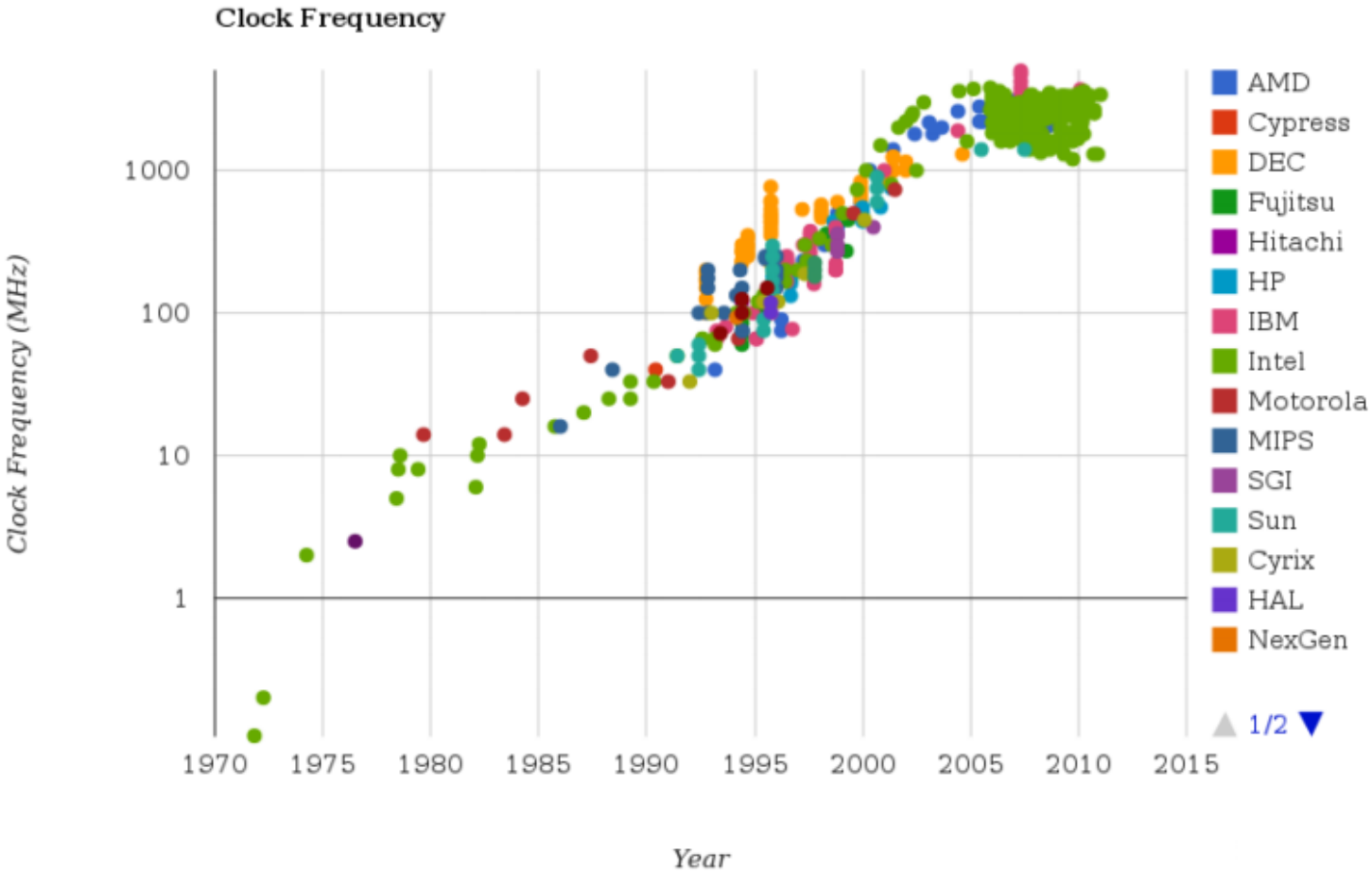
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Spis treści

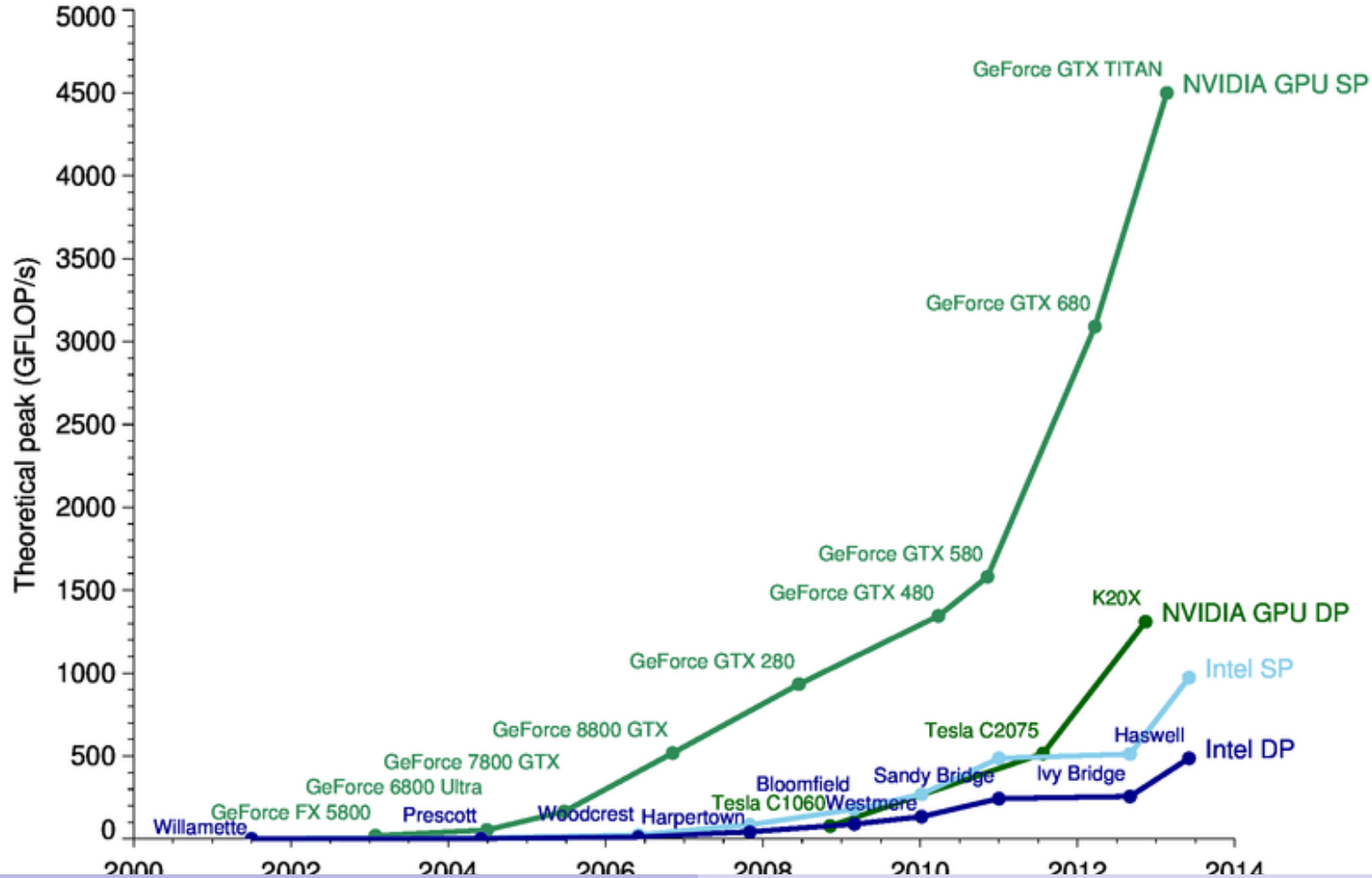
Zmniejszanie tranzystorów



Częstotliwość zegara



CPU vs GPU


Theoretical peak (GFLOP/s)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
2000
2002
2004
2006
2008
2010
2012
2014
GeForce GTX TITAN
NVIDIA GPU SP
GeForce GTX 680
GeForce GTX 580
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Willamette
Prescott
Woodcrest
Harpertown
Bloomfield
Tesla C1060
Westmere
Sandy Bridge
Ivy Bridge
Haswell
Tesla C2075
K20X
NVIDIA GPU DP
Intel SP
Intel DP

Superkomputery

TOP3 - 2010

Supercomputer	Max speed (TFLOPS)	Processors	Power (kW)
Tianhe-1A	2,566	14,336 Intel Xeon CPUs, 7,168 Nvidia Tesla GPUs	4040.00
Jaguar	1,759	224,256 AMD Opteron CPUs	6950.60
Nebulae	1,271	9,280 Intel Xeon CPUs, 4,640 Nvidia Tesla GPUs	2580.00

Źródło - 'OpenCL in Action',M.Scarpino

2009 - jeden na 20 najlepszych superkomputerów posiadał GPU

2012 - 62 z 500 najlepszych używają GPU,w tym najszybszy

Superkomputery

TOP3 - 2010

Supercomputer	Max speed (TFLOPS)	Processors	Power (kW)
Tianhe-1A	2,566	14,336 Intel Xeon CPUs, 7,168 Nvidia Tesla GPUs	4040.00
Jaguar	1,759	224,256 AMD Opteron CPUs	6950.60
Nebulae	1,271	9,280 Intel Xeon CPUs, 4,640 Nvidia Tesla GPUs	2580.00

Źródło - 'OpenCL in Action',M.Scarpino

2009 - jeden na 20 najlepszych superkomputerów posiadał GPU

2012 - 62 z 500 najlepszych używają GPU,w tym najszybszy

Superkomputery

TOP3 - 2010

Supercomputer	Max speed (TFLOPS)	Processors	Power (kW)
Tianhe-1A	2,566	14,336 Intel Xeon CPUs, 7,168 Nvidia Tesla GPUs	4040.00
Jaguar	1,759	224,256 AMD Opteron CPUs	6950.60
Nebulae	1,271	9,280 Intel Xeon CPUs, 4,640 Nvidia Tesla GPUs	2580.00

Źródło - 'OpenCL in Action',M.Scarpino

2009 - jeden na 20 najlepszych superkomputerów posiadał GPU

2012 - 62 z 500 najlepszych używają GPU,w tym najszybszy

GPGPU

General-Purpose computing on Graphics Processor Units

- 2006 - CUDA, NVIDIA
- 2009 - OpenCL, Khronos Group
- idealne zastosowanie? dużo danych, duża równoległość, mało zależności

GPGPU

General-Purpose computing on Graphics Processor Units

- 2006 - CUDA, NVIDIA
- 2009 - OpenCL, Khronos Group
- idealne zastosowanie? dużo danych, duża równoległość, mało zależności

GPGPU

General-Purpose computing on Graphics Processor Units

- 2006 - CUDA, NVIDIA
- 2009 - OpenCL, Khronos Group
- idealne zastosowanie? dużo danych, duża równoległość, mało zależności

GPGPU

General-Purpose computing on Graphics Processor Units

- 2006 - CUDA, NVIDIA
- 2009 - OpenCL, Khronos Group
- idealne zastosowanie? dużo danych, duża równoległość, mało zależności

GPGPU

General-Purpose computing on Graphics Processor Units

- 2006 - CUDA, NVIDIA
- 2009 - OpenCL, Khronos Group
- idealne zastosowanie? dużo danych, duża równoległość, mało zależności

SIMD

SIMD - Single Instruction, Multiple Data

- Cray-1 - superkomputer z procesorem wektorowy, 1976
- Cell(IBM, Toshiba, Sony) - jeden procesor skalarny, osiem wektorowych, 2000
- MMX/SSE/AVX - rozszerzenia wektorowe architektury x86

SIMD

SIMD - Single Instruction, Multiple Data

- Cray-1 - superkomputer z procesorem wektorowy, 1976
- Cell(IBM, Toshiba, Sony) - jeden procesor skalarny, osiem wektorowych, 2000
- MMX/SSE/AVX - rozszerzenia wektorowe architektury x86

SIMD

SIMD - Single Instruction, Multiple Data

- Cray-1 - superkomputer z procesorem wektorowy, 1976
- Cell(IBM, Toshiba, Sony) - jeden procesor skalarny, osiem wektorowych, 2000
- MMX/SSE/AVX - rozszerzenia wektorowe architektury x86

SIMD

SIMD - Single Instruction, Multiple Data

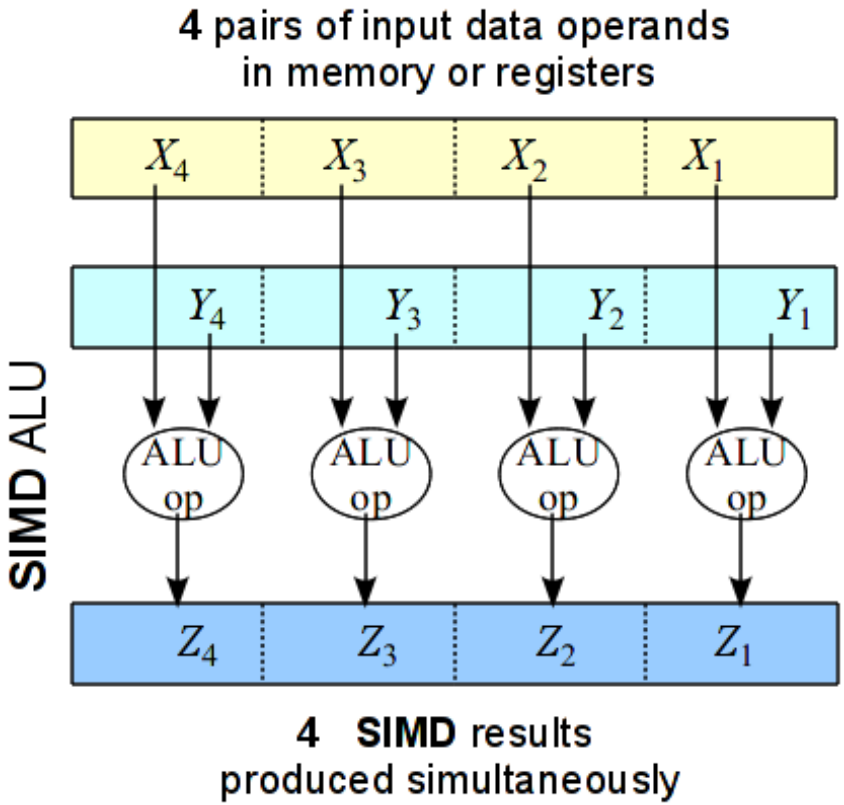
- Cray-1 - superkomputer z procesorem wektorowy, 1976
- Cell(IBM, Toshiba, Sony) - jeden procesor skalarny, osiem wektorowych, 2000
- MMX/SSE/AVX - rozszerzenia wektorowe architektury x86

SIMD

SIMD - Single Instruction, Multiple Data

- Cray-1 - superkomputer z procesorem wektorowy, 1976
- Cell(IBM, Toshiba, Sony) - jeden procesor skalarny, osiem wektorowych, 2000
- MMX/SSE/AVX - rozszerzenia wektorowe architektury x86

Rozszerzenia wektorowe



SIMT

SIMD - Single Instruction, Multiple Data
SIMT - Single Instruction, Multiple Thread

Dlaczego nowe pojęcie?

- marketing Nvidii?
- różnica względem np. SSE
- SPMD - Single Program, Multiple Data ('Patterns for Parallel Programming')

SIMT

SIMD - Single Instruction, Multiple Data
SIMT - Single Instruction, Multiple Thread
Dlaczego nowe pojęcie?

- marketing Nvidii?
- różnica względem np. SSE
- SPMD - Single Program, Multiple Data ('Patterns for Parallel Programming')

SIMT

SIMD - Single Instruction, Multiple Data
SIMT - Single Instruction, Multiple Thread
Dlaczego nowe pojęcie?

- marketing Nvidii?
- różnica względem np. SSE
- SPMD - Single Program, Multiple Data ('Patterns for Parallel Programming')

SIMT

SIMD - Single Instruction, Multiple Data
SIMT - Single Instruction, Multiple Thread
Dlaczego nowe pojęcie?

- marketing Nvidii?
- różnica względem np. SSE
- SPMD - Single Program, Multiple Data ('Patterns for Parallel Programming')

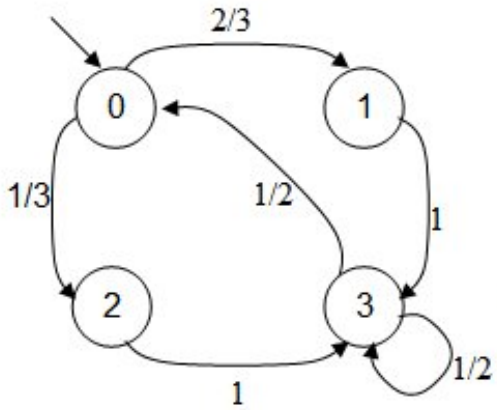
SIMT

SIMD - Single Instruction, Multiple Data
SIMT - Single Instruction, Multiple Thread
Dlaczego nowe pojęcie?

- marketing Nvidii?
- różnica względem np. SSE
- SPMD - Single Program, Multiple Data ('Patterns for Parallel Programming')

DTMC/CTMC

Łańcuch Markowa wraz z macierzą przejść



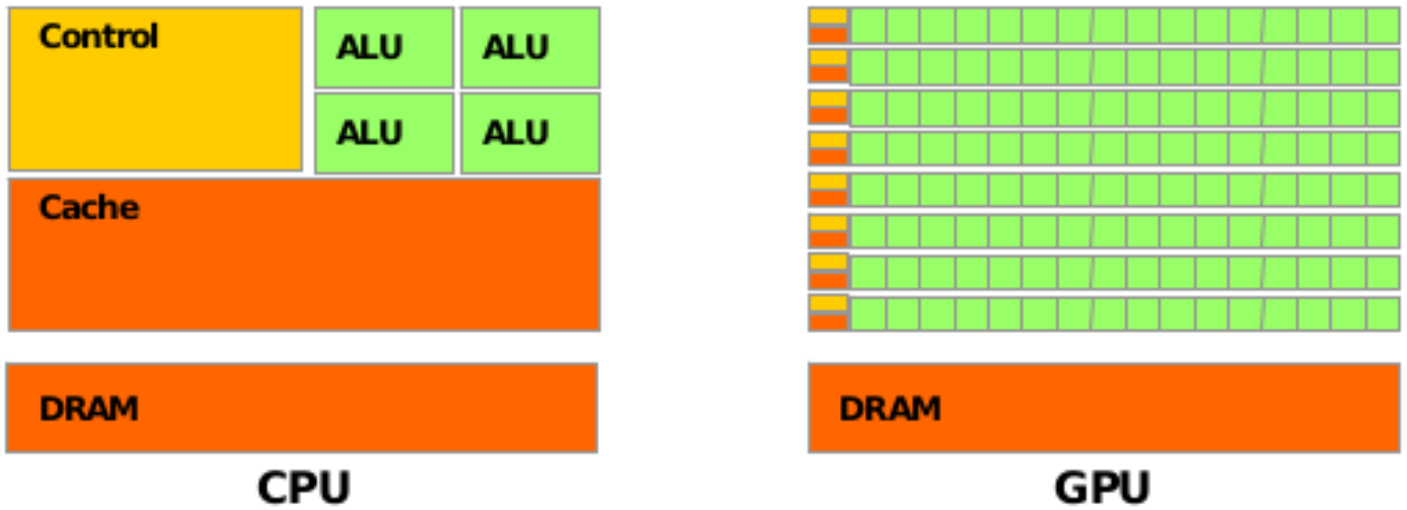
Matrix

$$\begin{pmatrix} 0 & 2/3 & 1/3 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1/2 & 0 & 0 & 1/2 \end{pmatrix}$$

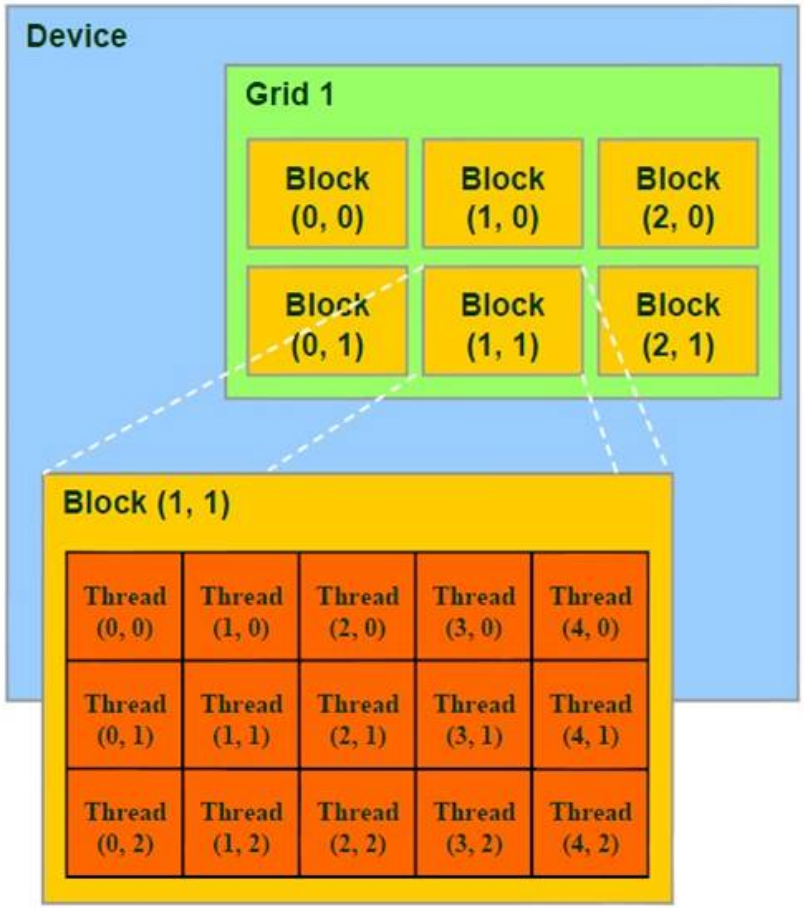
Probability distribution

Start.	(1,	0,	0,	0)
1.	(0,	2/3,	1/3,	0)
2.	(0,	0,	0,	1)
3.	(1/2,	0,	0,	1/2)
4.	(1/4,	1/3,	1/6,	1/4)

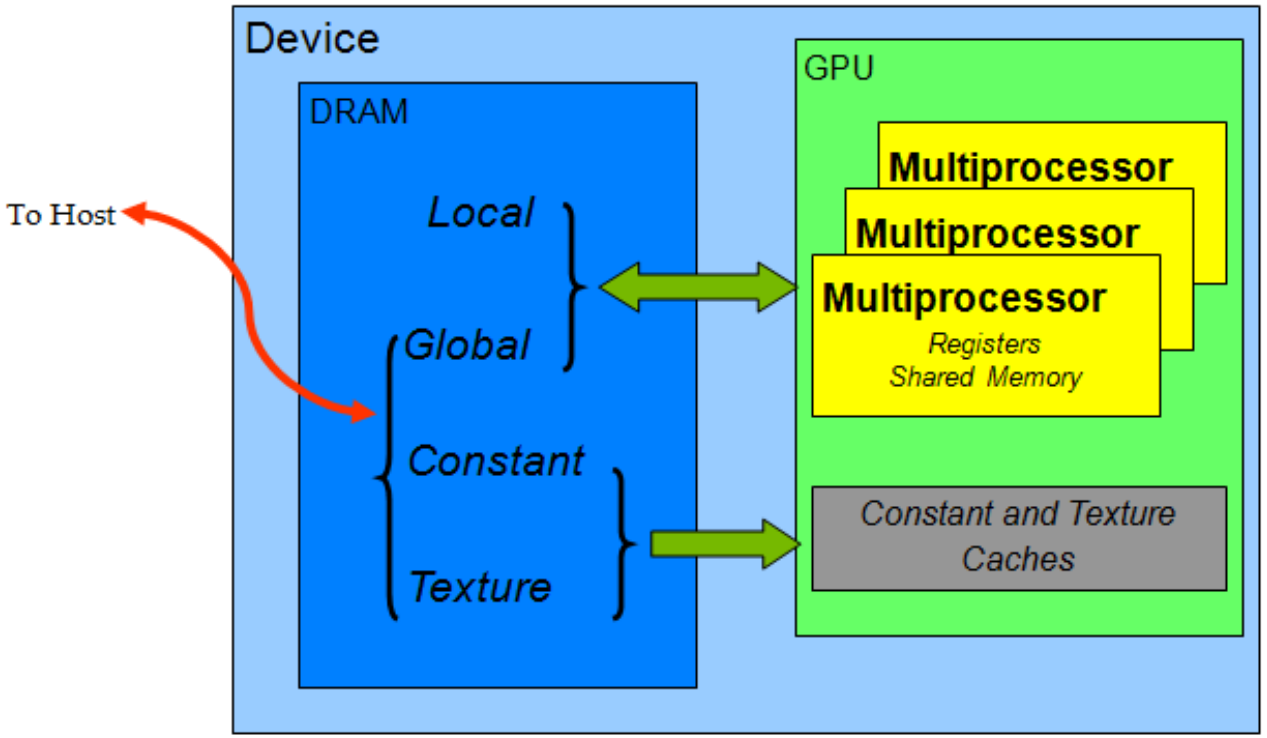
Control-intensive vs data-intensive



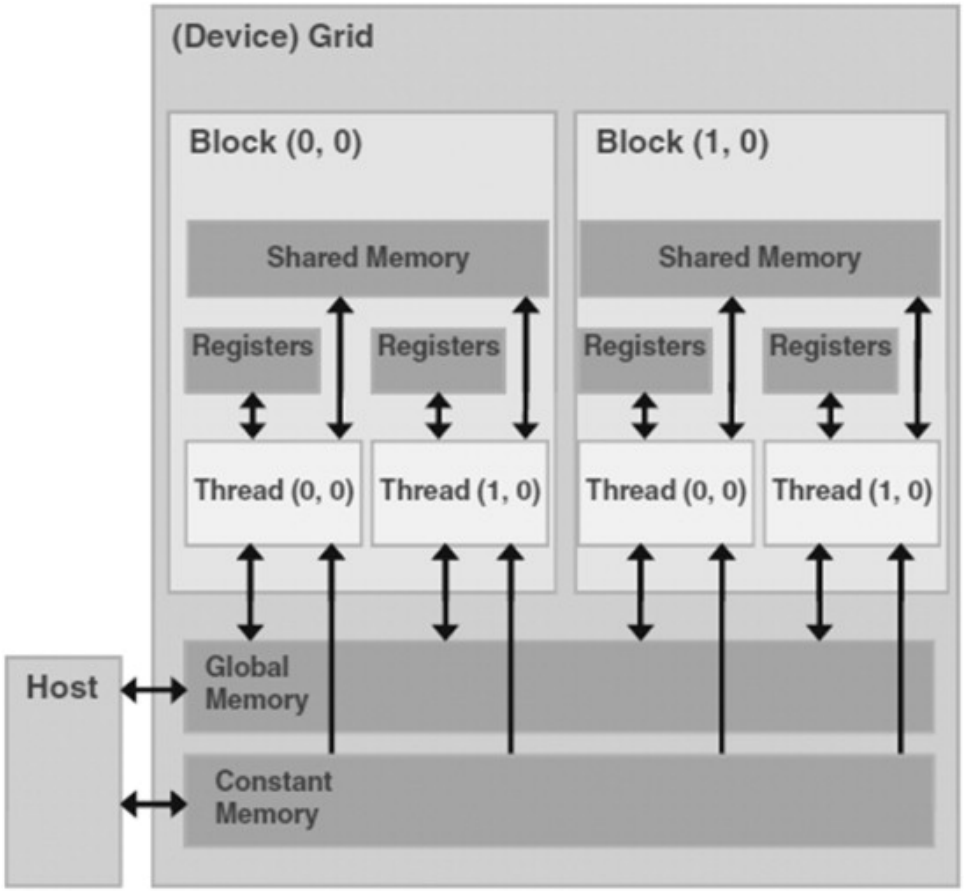
Siatka i wątki



Architektura urządzenia



Bloki na urządzeniu



Realizacja wątków na urządzeniu

- programista określa wątki(kernel), przydziela wątki do bloków, a bloki organizuje w siatkę
- CUDA przydziela bloki do multiprocesorów strumieniowych (Streaming Multiprocessor)
- SM przydziela zasoby dla bloku - warpy i pamięć współdzieloną
- SM wykonuje warpy, wybierając te aktywne z wszystkich dostępnych bloków
- synchronizacja tylko między wątkami w warpie!

Realizacja wątków na urządzeniu

- programista określa wątki(kernel), przydziela wątki do bloków, a bloki organizuje w siatkę
- CUDA przydziela bloki do multiprocesorów strumieniowych (Streaming Multiprocessor)
- SM przydziela zasoby dla bloku - warpy i pamięć współdzieloną
- SM wykonuje warpy, wybierając te aktywne z wszystkich dostępnych bloków
- synchronizacja tylko między wątkami w warpie!

Realizacja wątków na urządzeniu

- programista określa wątki(kernel), przydziela wątki do bloków, a bloki organizuje w siatkę
- CUDA przydziela bloki do multiprocesorów strumieniowych (Streaming Multiprocessor)
- SM przydziela zasoby dla bloku - warpy i pamięć współdzieloną
- SM wykonuje warpy, wybierając te aktywne z wszystkich dostępnych bloków
- synchronizacja tylko między wątkami w warpie!

Realizacja wątków na urządzeniu

- programista określa wątki(kernel), przydziela wątki do bloków, a bloki organizuje w siatkę
- CUDA przydziela bloki do multiprocesorów strumieniowych (Streaming Multiprocessor)
- SM przydziela zasoby dla bloku - warpy i pamięć współdzieloną
- SM wykonuje warpy, wybierając te aktywne z wszystkich dostępnych bloków
- synchronizacja tylko między wątkami w warpie!

Realizacja wątków na urządzeniu

- programista określa wątki(kernel), przydziela wątki do bloków, a bloki organizuje w siatkę
- CUDA przydziela bloki do multiprocesorów strumieniowych (Streaming Multiprocessor)
- SM przydziela zasoby dla bloku - warpy i pamięć współdzieloną
- SM wykonuje warpy, wybierając te aktywne z wszystkich dostępnych bloków
- synchronizacja tylko między wątkami w warpie!

Pamięć

Rodzaje pamięci:

- **rejestry(registers)** - 32-bitowe, dla wątku, 'on-chip',R/W
- **lokalna(local)** - dla wątku, 'off-chip',R/W
- **współdzielona(shared)** - dla bloku, 'on-chip',R/W
- **globalna(global)** - dla urządzenia, 'off-chip',R/W
- **stała(constant)** - dla urządzenia, 'off-chip',R

Pamięć

Rodzaje pamięci:

- **rejestry(registers)** - 32-bitowe, dla wątku, 'on-chip',R/W
- **lokalna(local)** - dla wątku, 'off-chip',R/W
- **współdzielona(shared)** - dla bloku, 'on-chip',R/W
- **globalna(global)** - dla urządzenia, 'off-chip',R/W
- **stała(constant)** - dla urządzenia, 'off-chip',R

Pamięć

Rodzaje pamięci:

- **rejestry(registers)** - 32-bitowe, dla wątku, 'on-chip',R/W
- **lokalna(local)** - dla wątku, 'off-chip',R/W
- **współdzielona(shared)** - dla bloku, 'on-chip',R/W
- **globalna(global)** - dla urządzenia, 'off-chip',R/W
- **stała(constant)** - dla urządzenia, 'off-chip',R

Pamięć

Rodzaje pamięci:

- **rejestry(registers)** - 32-bitowe, dla wątku, 'on-chip',R/W
- **lokalna(local)** - dla wątku, 'off-chip',R/W
- **współdzielona(shared)** - dla bloku, 'on-chip',R/W
- **globalna(global)** - dla urządzenia, 'off-chip',R/W
- **stała(constant)** - dla urządzenia, 'off-chip',R

Pamięć

Rodzaje pamięci:

- **rejestry(registers)** - 32-bitowe, dla wątku, 'on-chip',R/W
- **lokalna(local)** - dla wątku, 'off-chip',R/W
- **współdzielona(shared)** - dla bloku, 'on-chip',R/W
- **globalna(global)** - dla urządzenia, 'off-chip',R/W
- **stała(constant)** - dla urządzenia, 'off-chip',R

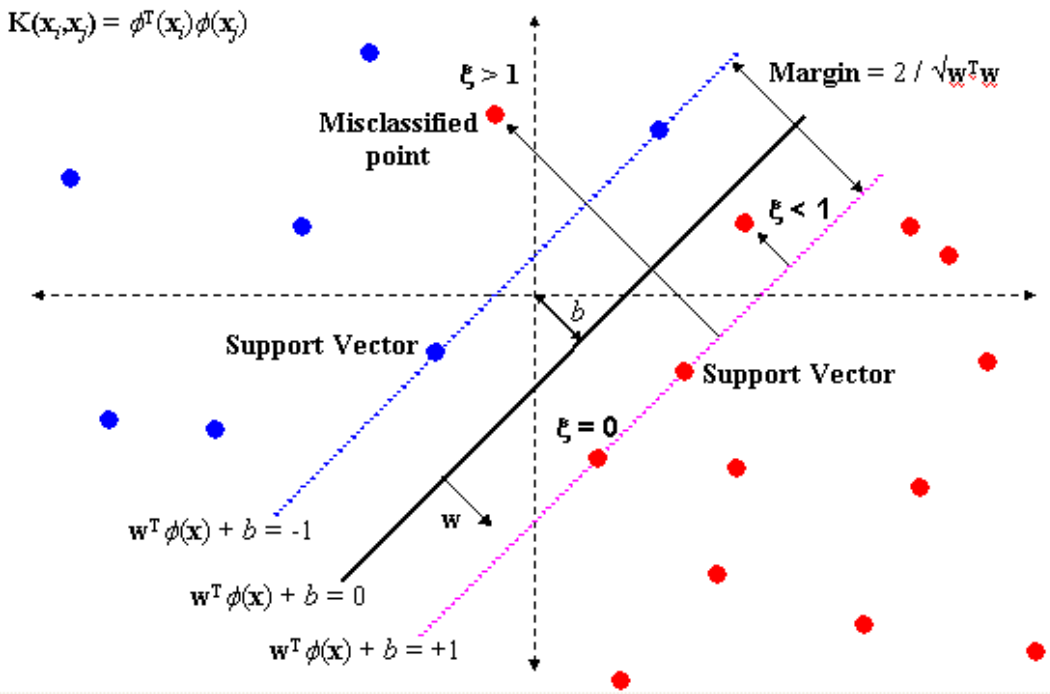
Pamięć

Rodzaje pamięci:

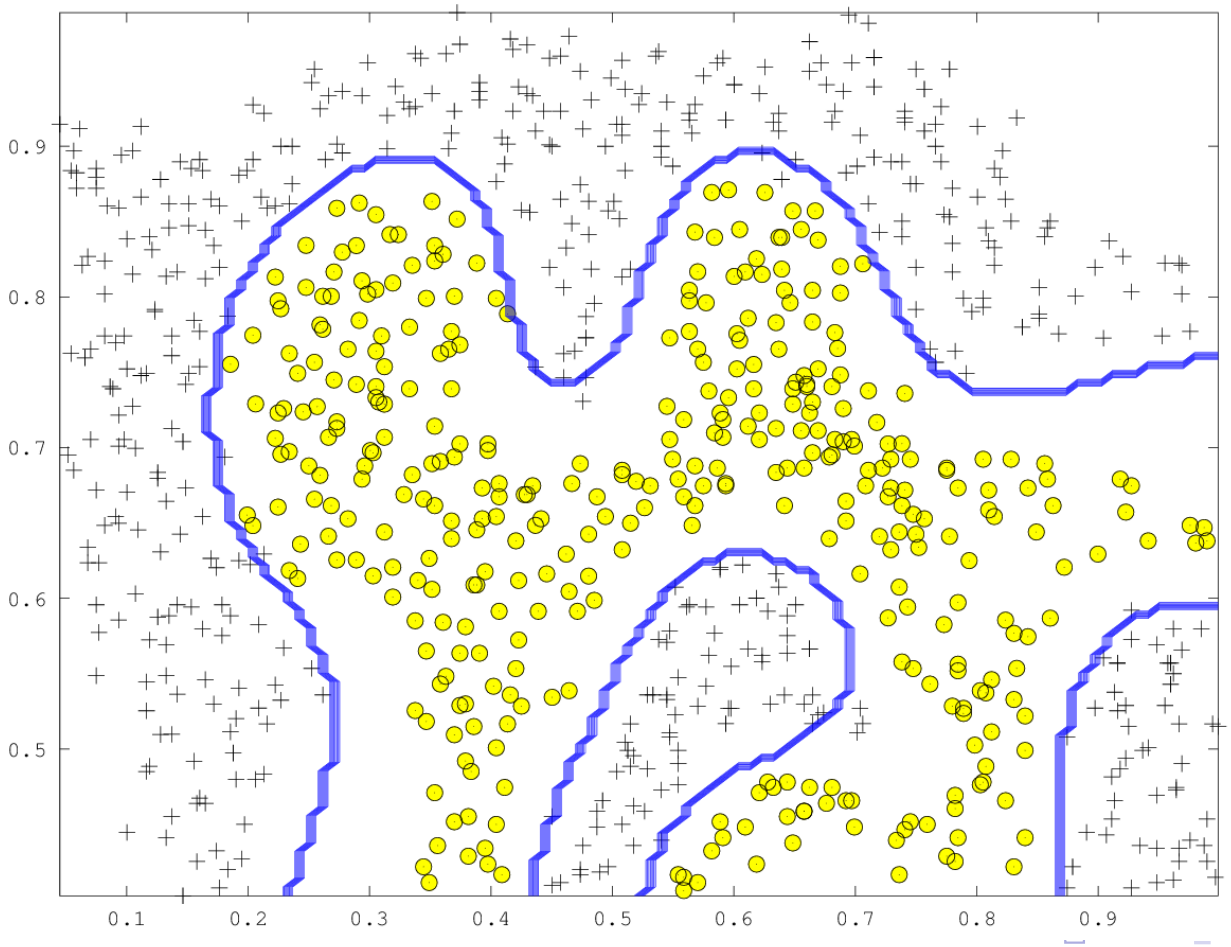
- **rejestry(registers)** - 32-bitowe, dla wątku, 'on-chip',R/W
- **lokalna(local)** - dla wątku, 'off-chip',R/W
- **współdzielona(shared)** - dla bloku, 'on-chip',R/W
- **globalna(global)** - dla urządzenia, 'off-chip',R/W
- **stała(constant)** - dla urządzenia, 'off-chip',R

Support Vector Machine

Linear SVM – Vladimir Napkin, 1963
Nonlinear SVM – Napkin,Guyon,Boser 1992



Support Vector Machine



Sequential Minimal Optimization

- 'Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines'
  John Platt, Microsoft Research, 1998
- 'Parallel sequential minimal optimization for the training of support vector machines' Cao L.J., Keerthi S. et al., 2006
- 'Fast Support Vector Machine Training and Classification on Graphics Processors' Catanzaro et al., 2008

Sequential Minimal Optimization

- 'Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines'
  John Platt, Microsoft Research, 1998
- 'Parallel sequential minimal optimization for the training of support vector machines' Cao L.J., Keerthi S. et al., 2006
- 'Fast Support Vector Machine Training and Classification on Graphics Processors' Catanzaro et al., 2008

Sequential Minimal Optimization

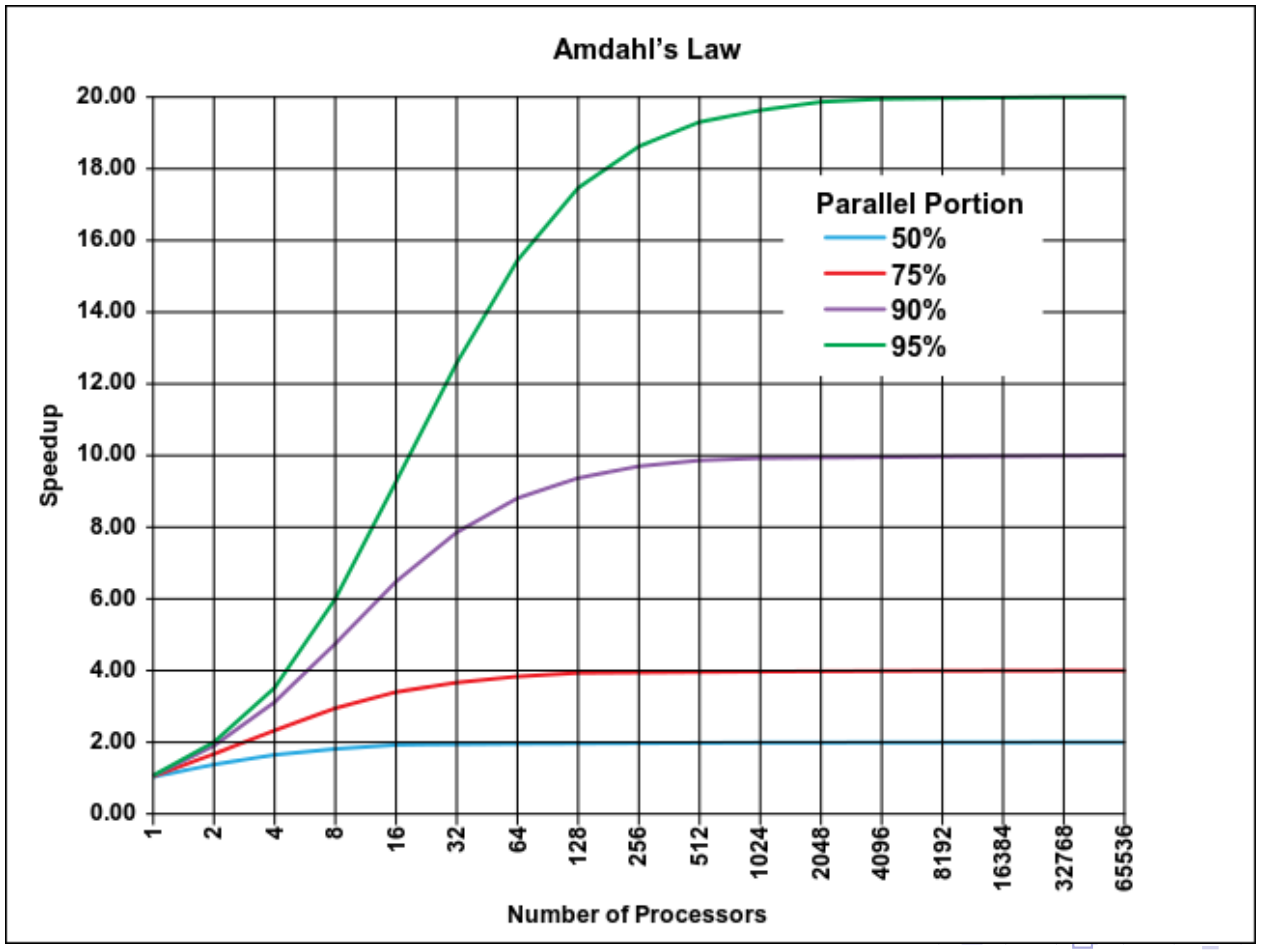
- 'Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines'
  John Platt, Microsoft Research, 1998
- 'Parallel sequential minimal optimization for the training of support vector machines' Cao L.J., Keerthi S. et al., 2006
- 'Fast Support Vector Machine Training and Classification on Graphics Processors' Catanzaro et al., 2008

Prawo Amdahla

$$S = \frac{1}{(1 - P) + \frac{P}{N}}$$

proporcja P tych obliczeń,
które mogą zostać zrównoleglone na N jednostkach wykonawczych

Prawo Amdahla



Opóźnienia

- pamięć globalna jest wolna!
- pamięć współdzielona ma ok. 100 razy mniejsze opóźnienie
- 'coalescing' dla pamięci globalnej
- 'bank conflicts' dla pamięci współdzielonej
- liczba rejestrów nie jest nieskończona!
- w nowszych urządzeniach(Kepler) dodatkowe 48 kB pamięci cache tylko do odczytu

Opóźnienia

- pamięć globalna jest wolna!
- pamięć współdzielona ma ok. 100 razy mniejsze opóźnienie
- 'coalescing' dla pamięci globalnej
- 'bank conflicts' dla pamięci współdzielonej
- liczba rejestrów nie jest nieskończona!
- w nowszych urządzeniach(Kepler) dodatkowe 48 kB pamięci cache tylko do odczytu

Opóźnienia

- pamięć globalna jest wolna!
- pamięć współdzielona ma ok. 100 razy mniejsze opóźnienie
- 'coalescing' dla pamięci globalnej
- 'bank conflicts' dla pamięci współdzielonej
- liczba rejestrów nie jest nieskończona!
- w nowszych urządzeniach(Kepler) dodatkowe 48 kB pamięci cache tylko do odczytu

Opóźnienia

- pamięć globalna jest wolna!
- pamięć współdzielona ma ok. 100 razy mniejsze opóźnienie
- 'coalescing' dla pamięci globalnej
- 'bank conflicts' dla pamięci wspóldzielonej
- liczba rejestrów nie jest nieskończona!
- w nowszych urządzeniach(Kepler) dodatkowe 48 kB pamięci cache tylko do odczytu

Opóźnienia

- pamięć globalna jest wolna!
- pamięć współdzielona ma ok. 100 razy mniejsze opóźnienie
- 'coalescing' dla pamięci globalnej
- 'bank conflicts' dla pamięci wspóldzielonej
- liczba rejestrów nie jest nieskończona!
- w nowszych urządzeniach(Kepler) dodatkowe 48 kB pamięci cache tylko do odczytu

Opóźnienia

- pamięć globalna jest wolna!
- pamięć współdzielona ma ok. 100 razy mniejsze opóźnienie
- 'coalescing' dla pamięci globalnej
- 'bank conflicts' dla pamięci wspóldzielonej
- liczba rejestrów nie jest nieskończona!
- w nowszych urządzeniach(Kepler) dodatkowe 48 kB pamięci cache tylko do odczytu

Rozgałęzienia

- jeden warp może wykonywać jedną instrukcję w danej chwili
- gdy następuje N rozgałęzień, to N-krotnie spada wydajność
- kompilator może dokonać optymalizacji
- rozdzielanie danych na warpy i kernele

Rozgałęzienia

- jeden warp może wykonywać jedną instrukcję w danej chwili
- gdy następuje N rozgałęzień, to N-krotnie spada wydajność
- kompilator może dokonać optymalizacji
- rozdzielanie danych na warpy i kernele

Rozgałęzienia

- jeden warp może wykonywać jedną instrukcję w danej chwili
- gdy następuje N rozgałęzień, to N-krotnie spada wydajność
- kompilator może dokonać optymalizacji
- rozdzielanie danych na warpy i kernele

Rozgałęzienia

- jeden warp może wykonywać jedną instrukcję w danej chwili
- gdy następuje N rozgałęzień, to N-krotnie spada wydajność
- kompilator może dokonać optymalizacji
- rozdzielanie danych na warpy i kernele

Technicznie

- kod kernela w pliku .cu, podzbiór języka C - bez rekurencji, wskaźników itd.
- w nowszych wersjach wsparcie dla niektórych mechanizmów języka C++ (np. klasy bez funkcji wirtualnych)
- kompilator nvcc
- łączymy z aplikacją w C lub C++
- wrappery do Pythona, Javy, Matlaba....

Technicznie

- kod kernela w pliku .cu, podzbiór języka C - bez rekurencji, wskaźników itd.
- w nowszych wersjach wsparcie dla niektórych mechanizmów języka C++ (np. klasy bez funkcji wirtualnych)
- kompilator nvcc
- łączymy z aplikacją w C lub C++
- wrappery do Pythona, Javy, Matlaba....

Technicznie

- kod kernela w pliku .cu, podzbiór języka C - bez rekurencji, wskaźników itd.
- w nowszych wersjach wsparcie dla niektórych mechanizmów języka C++ (np. klasy bez funkcji wirtualnych)
- kompilator nvcc
- łączymy z aplikacją w C lub C++
- wrappery do Pythona, Javy, Matlaba....

Technicznie

- kod kernela w pliku .cu, podzbiór języka C - bez rekurencji, wskaźników itd.
- w nowszych wersjach wsparcie dla niektórych mechanizmów języka C++ (np. klasy bez funkcji wirtualnych)
- kompilator nvcc
- łączymy z aplikacją w C lub C++
- wrappery do Pythona, Javy, Matlaba....

Technicznie

- kod kernela w pliku .cu, podzbiór języka C - bez rekurencji, wskaźników itd.
- w nowszych wersjach wsparcie dla niektórych mechanizmów języka C++ (np. klasy bez funkcji wirtualnych)
- kompilator nvcc
- łączymy z aplikacją w C lub C++
- wrappery do Pythona, Javy, Matlaba....

Cykl życia programu

- alokacja bloków pamięci na host i urządzeniu
- transfer danych na urządzenie
- wywołanie kernela
- transfer danych z urządzenia na host
- analiza wyników, ponowne użycie GPU

Cykl życia programu

- alokacja bloków pamięci na host i urządzeniu
- transfer danych na urządzenie
- wywołanie kernela
- transfer danych z urządzenia na host
- analiza wyników, ponowne użycie GPU

Cykl życia programu

- alokacja bloków pamięci na host i urządzeniu
- transfer danych na urządzenie
- wywołanie kernela
- transfer danych z urządzenia na host
- analiza wyników, ponowne użycie GPU

Cykl życia programu

- alokacja bloków pamięci na host i urządzeniu
- transfer danych na urządzenie
- wywołanie kernela
- transfer danych z urządzenia na host
- analiza wyników, ponowne użycie GPU

Cykl życia programu

- alokacja bloków pamięci na host i urządzeniu
- transfer danych na urządzenie
- wywołanie kernela
- transfer danych z urządzenia na host
- analiza wyników, ponowne użycie GPU

Wsparcie dla CUDA

- cuFFT dla szybkiej transformaty Fouriera
- cuBLAS - dla algebry liniowej
- CUDA Math Library
- cuSPARSE dla macierzy rzadkich i wiele innych
- wsparcie dla CUDA w wielu projektach open-source: OpenCV, ICP

Wsparcie dla CUDA

- cuFFT dla szybkiej transformaty Fouriera
- cuBLAS - dla algebry liniowej
- CUDA Math Library
- cuSPARSE dla macierzy rzadkich i wiele innych
- wsparcie dla CUDA w wielu projektach open-source: OpenCV, ICP

Wsparcie dla CUDA

- cuFFT dla szybkiej transformaty Fouriera
- cuBLAS - dla algebry liniowej
- CUDA Math Library
- cuSPARSE dla macierzy rzadkich i wiele innych
- wsparcie dla CUDA w wielu projektach open-source: OpenCV, ICP

Wsparcie dla CUDA

- cuFFT dla szybkiej transformaty Fouriera
- cuBLAS - dla algebry liniowej
- CUDA Math Library
- cuSPARSE dla macierzy rzadkich i wiele innych
- wsparcie dla CUDA w wielu projektach open-source: OpenCV, ICP

Wsparcie dla CUDA

- cuFFT dla szybkiej transformaty Fouriera
- cuBLAS - dla algebry liniowej
- CUDA Math Library
- cuSPARSE dla macierzy rzadkich i wiele innych
- wsparcie dla CUDA w wielu projektach open-source: OpenCV, ICP

Rozwój CUDA

- 1.3 - wsparcie dla obliczeń zmiennoprzecinkowych podwójnej precyzji, 16 kB pamięci współdzielonej na multiprocesor, 16k rejestrów na multiprocesor
- 2.x - wsparcie dla C++, 48 kB pamięci współdzielonej na multiprocesor 32k rejestrów na multiprocesor
- 3.x,4.x,5.x - 64 kB pamięci współdzielonej na multiprocesor, dynamiczna równoległość i wiele innych

Rozwój CUDA

- 1.3 - wsparcie dla obliczeń zmiennoprzecinkowych podwójnej precyzji, 16 kB pamięci współdzielonej na multiprocesor, 16k rejestrów na multiprocesor
- 2.x - wsparcie dla C++, 48 kB pamięci współdzielonej na multiprocesor 32k rejestrów na multiprocesor
- 3.x,4.x,5.x - 64 kB pamięci współdzielonej na multiprocesor, dynamiczna równoległość i wiele innych

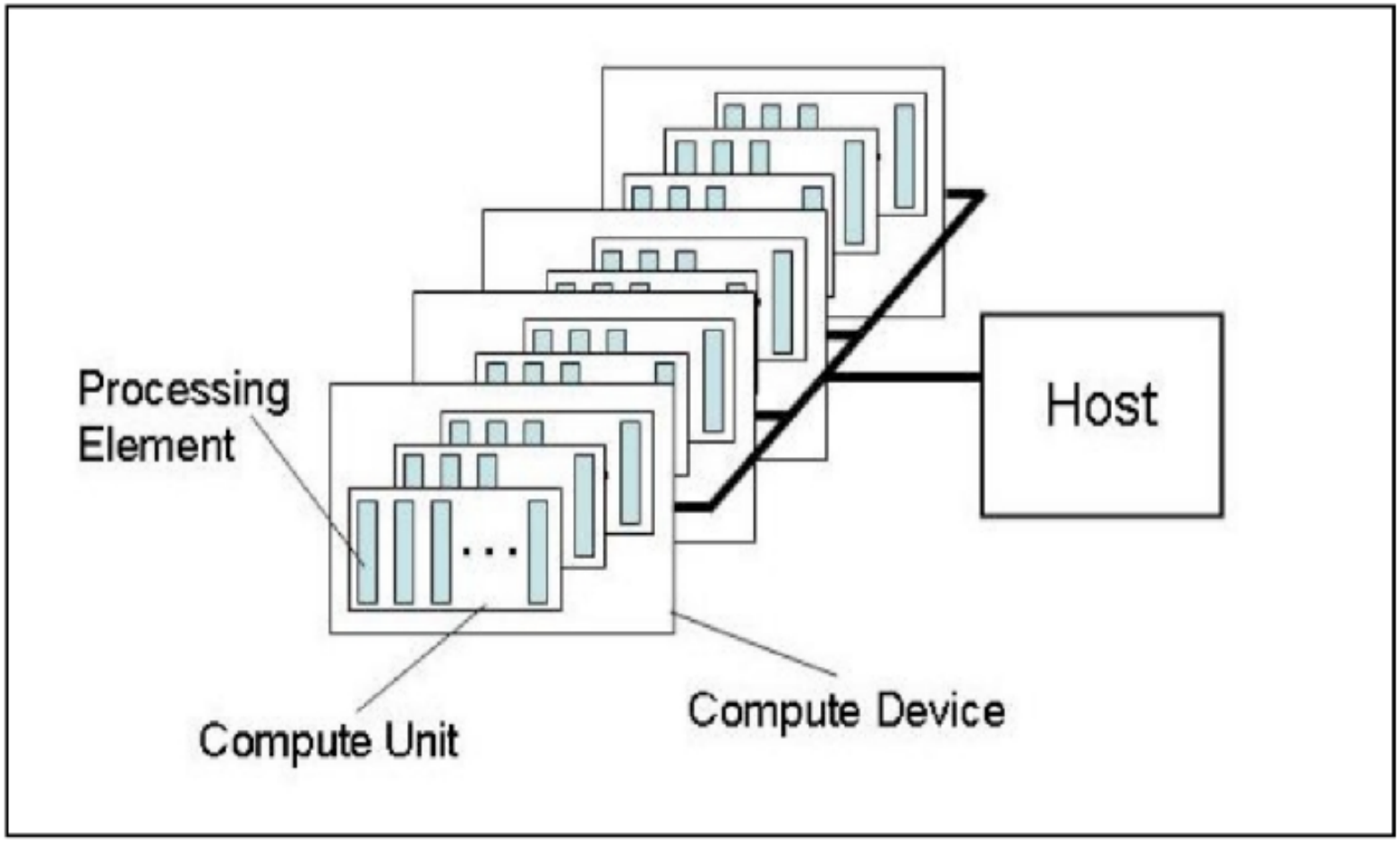
Rozwój CUDA

- 1.3 - wsparcie dla obliczeń zmiennoprzecinkowych podwójnej precyzji, 16 kB pamięci współdzielonej na multiprocesor, 16k rejestrów na multiprocesor
- 2.x - wsparcie dla C++, 48 kB pamięci współdzielonej na multiprocesor 32k rejestrów na multiprocesor
- 3.x,4.x,5.x - 64 kB pamięci współdzielonej na multiprocesor, dynamiczna równoległość i wiele innych

OpenCL


KHRONOS GROUP
OpenCL
3DLABS
ACTIVISION
BLIZZARD
AMD
ARM
BROADCOM
EA
codeplay
ERICSSON
freescale
IBM
intel
Imagination
MOTOROLA
mvidia
NOKIA
NVIDIA
Petapath
QNX
RAPID
S3 GRAPHICS
SAMSUNG
Seaweed SYSTEMS
TAKUMI
TEXAS INSTRUMENTS

Model urządzeń



Różnice

CUDA term	OpenCL term
GPU	Device
Multiprocessor	Compute Unit
Scalar core	Processing element
Global memory	Global memory
Shared (per-block) memory	Local memory
Local memory (automatic, or local)	Private memory
kernel	program
block	work-group
thread	work item

OpenCL context

- devices - np. GPU, CPU, DSP
- kernels
- program objects - coś na wzór dynamicznej biblioteki, zawiera funkcje używane przez kernel, budowane w locie
- memory objects

OpenCL context

- devices - np. GPU, CPU, DSP
- kernels
- program objects - coś na wzór dynamicznej biblioteki, zawiera funkcje używane przez kernel, budowane w locie
- memory objects

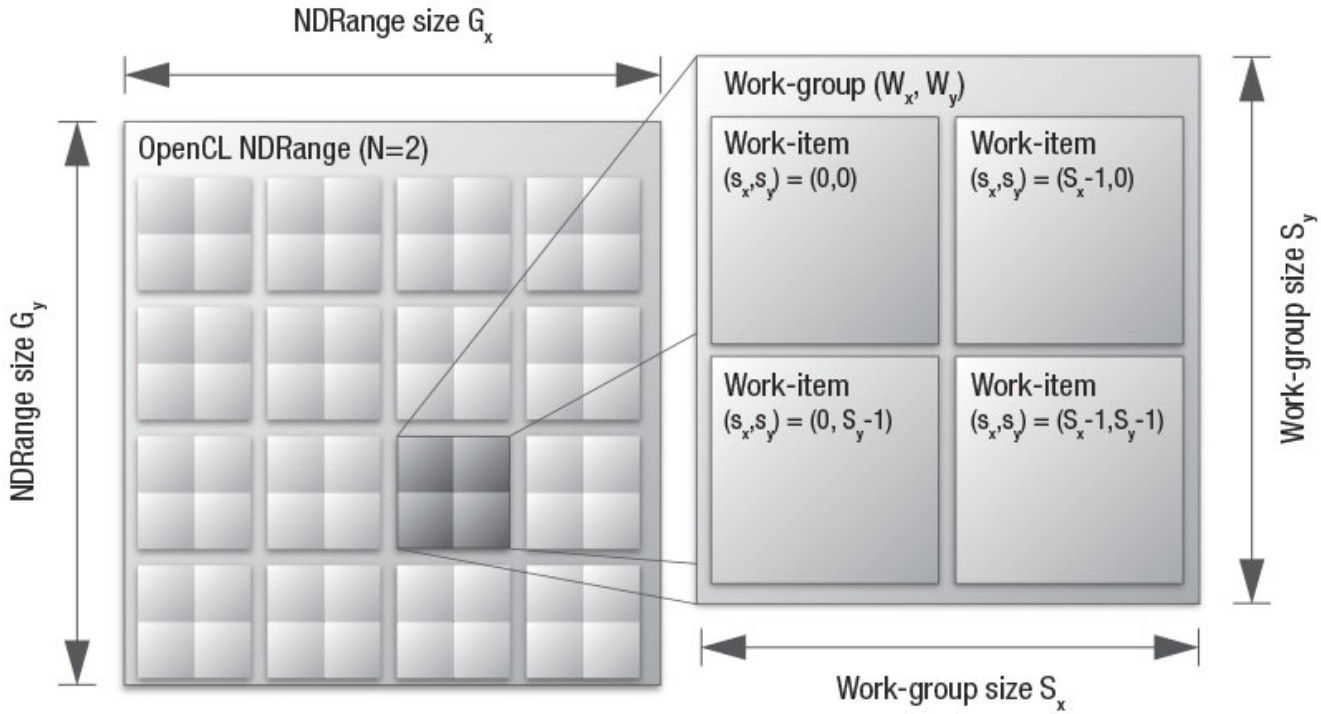
OpenCL context

- devices - np. GPU, CPU, DSP
- kernels
- program objects - coś na wzór dynamicznej biblioteki, zawiera funkcje używane przez kernel, budowane w locie
- memory objects

OpenCL context

- devices - np. GPU, CPU, DSP
- kernels
- program objects - coś na wzór dynamicznej biblioteki, zawiera funkcje używane przez kernel, budowane w locie
- memory objects

Work-group i work-items



Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Cykl życia programu

- wybranie platformy OpenCL
- wybranie urządzenia dostępnego na platformie i utworzenie kontekstu wykonania
- utworzenie kolejki zadań
- kompilacja kernela(w trakcie działania programu hosta!)
- alokacja buforów danych na host i urządzeniu
- wrzucenie do kolejki zadań transferu danych na urządzenie
- wrzucenie do kolejki zadania wywołania kernela
- wrzucenie do kolejki zadań transferu danych z urządzenia na host
- analiza wyników, dalsza praca programu

Rozwój OpenCL

- 1.0 - grudzień 2008, wsparcie AMD, obietnica wsparcia NVIDII
- 1.1 - czerwiec 2010, poprawa funkcjonalności i elastyczności
- 1.2 - listopad 2011, poprawa wydajności
- 2.0 - lipiec 2013, dynamiczna równoległość i inne
- propozycje wprowadzenia automatycznego zrównoleglania do Javy 9(2016?) przy użyciu OpenCL

Rozwój OpenCL

- 1.0 - grudzień 2008, wsparcie AMD, obietnica wsparcia NVIDII
- 1.1 - czerwiec 2010, poprawa funkcjonalności i elastyczności
- 1.2 - listopad 2011, poprawa wydajności
- 2.0 - lipiec 2013, dynamiczna równoległość i inne
- propozycje wprowadzenia automatycznego zrównoleglania do Javy 9(2016?) przy użyciu OpenCL

Rozwój OpenCL

- 1.0 - grudzień 2008, wsparcie AMD, obietnica wsparcia NVIDII
- 1.1 - czerwiec 2010, poprawa funkcjonalności i elastyczności
- 1.2 - listopad 2011, poprawa wydajności
- 2.0 - lipiec 2013, dynamiczna równoległość i inne
- propozycje wprowadzenia automatycznego zrównoleglania do Javy 9(2016?) przy użyciu OpenCL

Rozwój OpenCL

- 1.0 - grudzień 2008, wsparcie AMD, obietnica wsparcia NVIDII
- 1.1 - czerwiec 2010, poprawa funkcjonalności i elastyczności
- 1.2 - listopad 2011, poprawa wydajności
- 2.0 - lipiec 2013, dynamiczna równoległość i inne
- propozycje wprowadzenia automatycznego zrównoleglania do Javy 9(2016?) przy użyciu OpenCL

Rozwój OpenCL

- 1.0 - grudzień 2008, wsparcie AMD, obietnica wsparcia NVIDII
- 1.1 - czerwiec 2010, poprawa funkcjonalności i elastyczności
- 1.2 - listopad 2011, poprawa wydajności
- 2.0 - lipiec 2013, dynamiczna równoległość i inne
- propozycje wprowadzenia automatycznego zrównoleglania do Javy 9(2016?) przy użyciu OpenCL

Python

Skąd popularność Pythona?

- czysta, czytelna i piękna składnia
- dynamicznie typowany, interpretowany
- ogrom bibliotek i narzędzi
- dobra dokumentacja i silne wsparcie społeczności
- wysoka produktywność

Skąd popularność Pythona?

- czysta, czytelna i piękna składnia
- dynamicznie typowany, interpretowany
- ogrom bibliotek i narzędzi
- dobra dokumentacja i silne wsparcie społeczności
- wysoka produktywność

Skąd popularność Pythona?

- czysta, czytelna i piękna składnia
- dynamicznie typowany, interpretowany
- ogrom bibliotek i narzędzi
- dobra dokumentacja i silne wsparcie społeczności
- wysoka produktywność

Skąd popularność Pythona?

- czysta, czytelna i piękna składnia
- dynamicznie typowany, interpretowany
- ogrom bibliotek i narzędzi
- dobra dokumentacja i silne wsparcie społeczności
- wysoka produktywność

Skąd popularność Pythona?

- czysta, czytelna i piękna składnia
- dynamicznie typowany, interpretowany
- ogrom bibliotek i narzędzi
- dobra dokumentacja i silne wsparcie społeczności
- wysoka produktywność

Skąd popularność Pythona?

- czysta, czytelna i piękna składnia
- dynamicznie typowany, interpretowany
- ogrom bibliotek i narzędzi
- dobra dokumentacja i silne wsparcie społeczności
- wysoka produktywność

Uruchamianie funkcji na GPU

przenośność!

```
import numpy as np
from numbapro import vectorize

@vectorize(['float32(float32, float32)'], target='gpu')
def Add(a, b):
  return a + b

# Initialize arrays
A = np.ones(N, dtype=np.float32)
B = np.ones(A.shape, dtype=A.dtype)
C = np.empty_like(A, dtype=A.dtype)

# Add arrays on GPU
C = Add(A, B)
```

Kernele CUDA w Pythonie

```
@cuda.jit(argtypes=[f8, f8, f8, f8, uint8[:,:], uint32])
def mandel_kernel(min_x, max_x, min_y, max_y, image, iters):
  height = image.shape[0]
  width = image.shape[1]

  pixel_size_x = (max_x - min_x) / width
  pixel_size_y = (max_y - min_y) / height

  startX = cuda.blockDim.x * cuda.blockIdx.x + cuda.ThreadIdx.x
  startY = cuda.blockDim.y * cuda.blockIdx.y + cuda.ThreadIdx.y
  gridX = cuda.gridDim.x * cuda.blockDim.x;
  gridY = cuda.gridDim.y * cuda.blockDim.y;

  for x in range(startX, width, gridX):
    real = min_x + x * pixel_size_x
    for y in range(startY, height, gridY):
      imag = min_y + y * pixel_size_y
      image[y, x] = mandel(real, imag, iters)
```

Oficjalne materiały

- 'CUDA C Programming Guide'
- 'CUDA C Best Practices Guide'
- 'OpenCL Programming Guide for the CUDA Architecture'
- 'OpenCL Best Practices Guide'

Oficjalne materiały

- 'CUDA C Programming Guide'
- 'CUDA C Best Practices Guide'
- 'OpenCL Programming Guide for the CUDA Architecture'
- 'OpenCL Best Practices Guide'

Oficjalne materiały

- 'CUDA C Programming Guide'
- 'CUDA C Best Practices Guide'
- 'OpenCL Programming Guide for the CUDA Architecture'
- 'OpenCL Best Practices Guide'

Oficjalne materiały

- 'CUDA C Programming Guide'
- 'CUDA C Best Practices Guide'
- 'OpenCL Programming Guide for the CUDA Architecture'
- 'OpenCL Best Practices Guide'

E-learning

- 'Coursera: Heterogeneous Parallel Programming, University of Illinois'
- 'Udacity: Introduction to Parallel Programming, University of California + NVIDIA'

E-learning

- 'Coursera: Heterogeneous Parallel Programming, University of Illinois'
- 'Udacity: Introduction to Parallel Programming, University of California + NVIDIA'

Książki

- 'CUDA by Example: An Introduction to General-Purpose GPU Programming'
- 'Heterogeneous Computing with OpenCL '
- 'CUDA Application Design and Development'

Książki

- 'CUDA by Example: An Introduction to General-Purpose GPU Programming'
- 'Heterogeneous Computing with OpenCL '
- 'CUDA Application Design and Development'

Książki

- 'CUDA by Example: An Introduction to General-Purpose GPU Programming'
- 'Heterogeneous Computing with OpenCL '
- 'CUDA Application Design and Development'