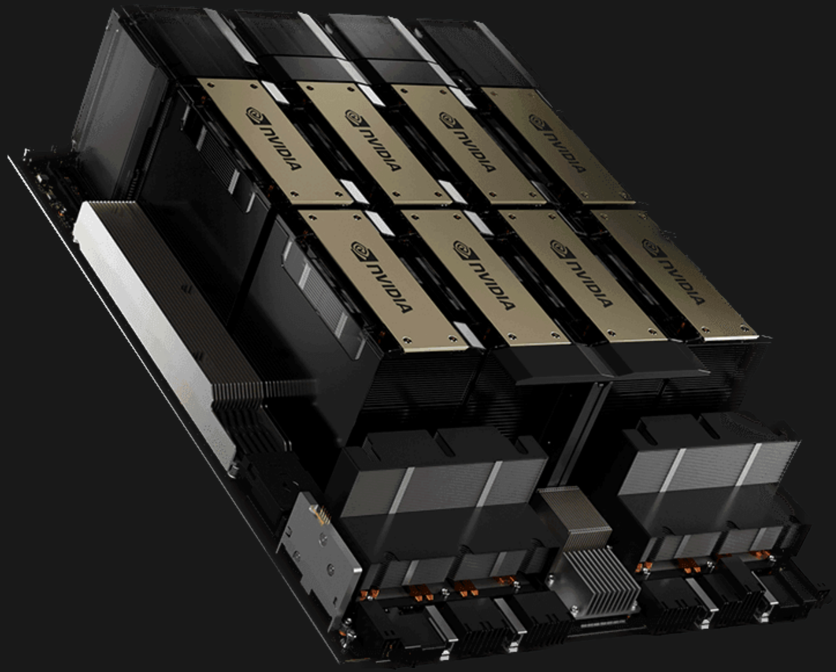
Introduction to GPU Programming

Yonglei Wang (ENCCS/NSC@LiU)

What is GPU?

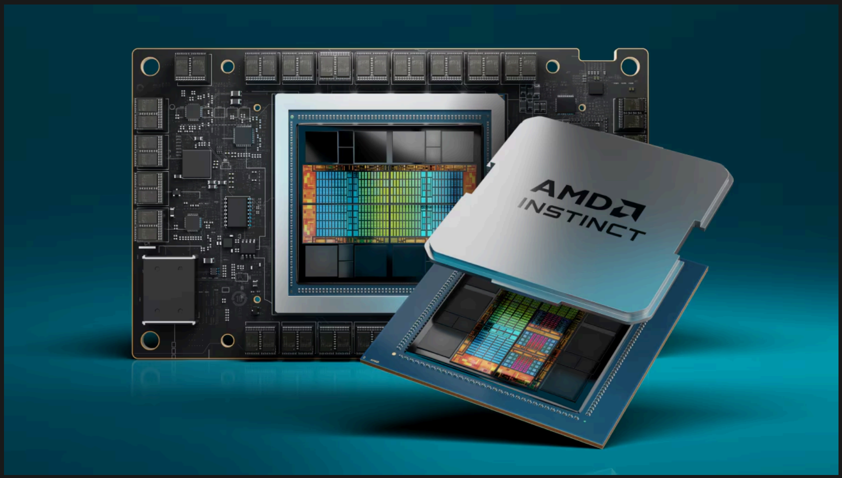
- GPU is a specialized electronic circuit.
- Originally it was developed for computer graphics and image processing.
- Now GPUs have evolved into general-proposed accelerators for massive parallel computing.

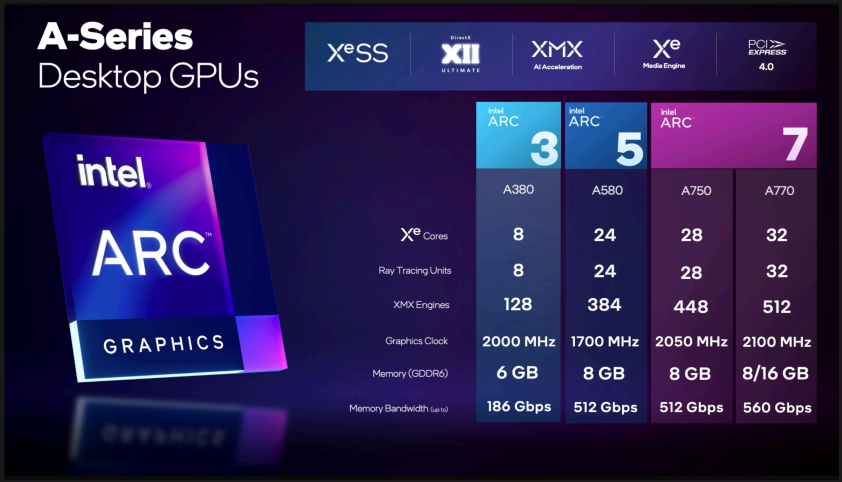
GPU cards



NVIDIA RTX 4090 & H200

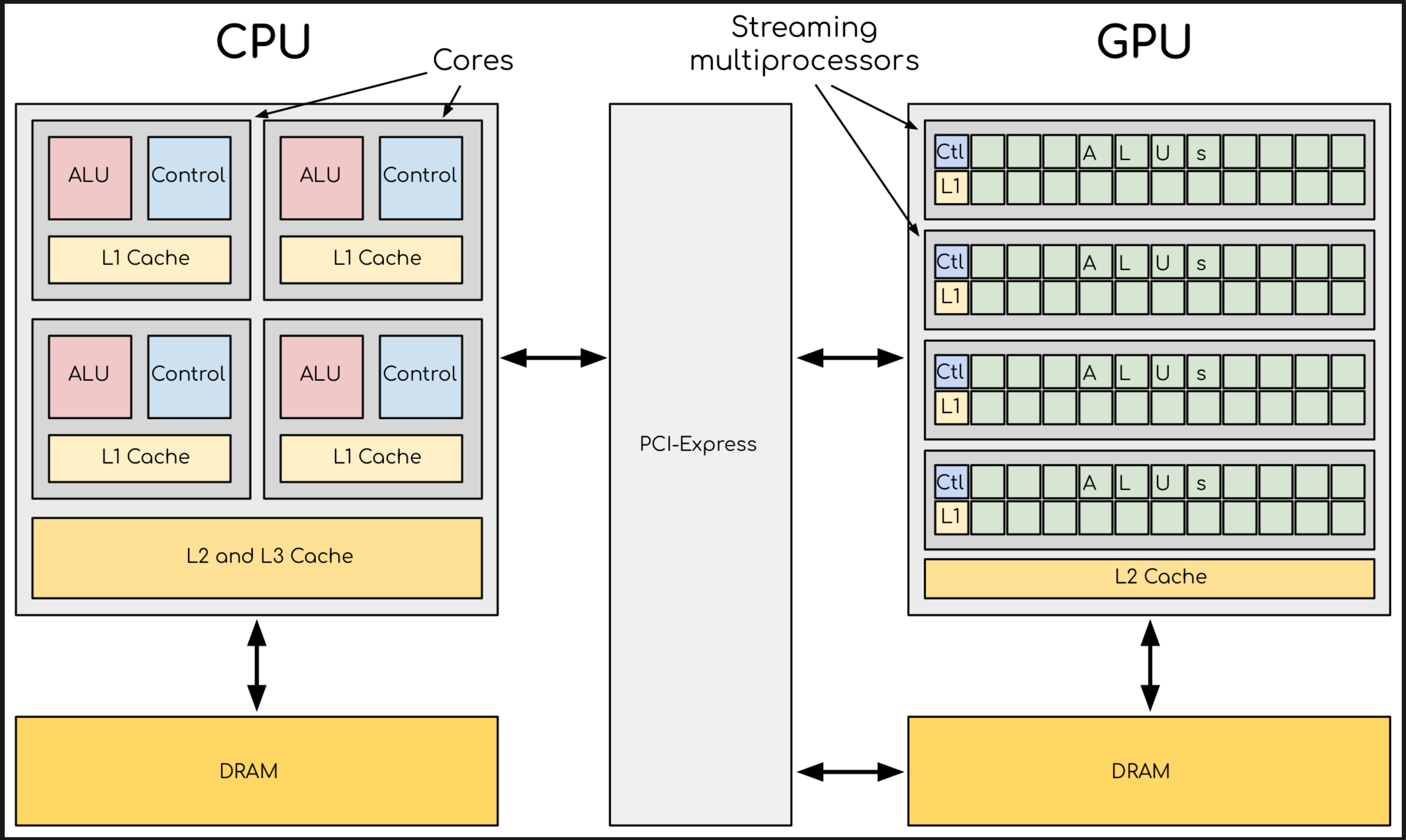
GPU cards





AMD MI300 series *vs.* Intel's ARC series

CPU vs. GPU



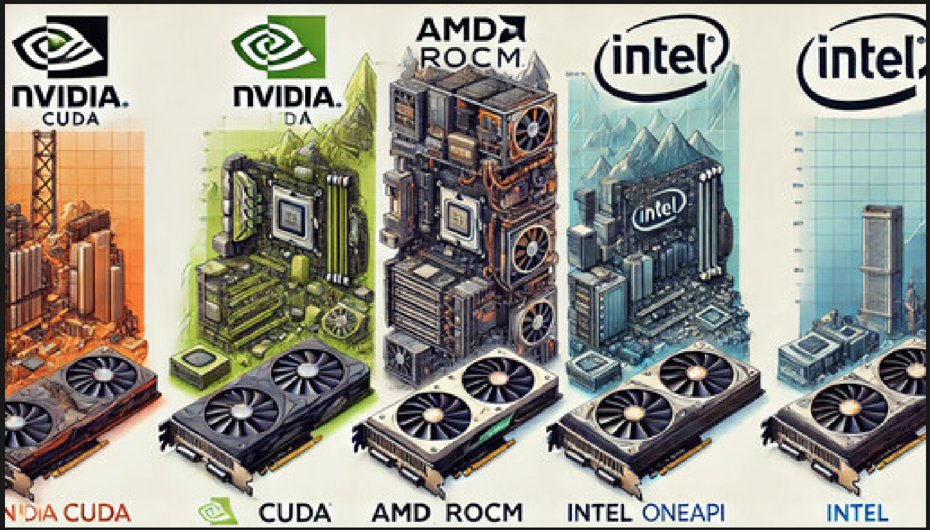
Ref: GPU Programming: When, Why and How?

Top 10 HPCs

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107
6	**Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	1,305,600	270.00	353.75	5,194
7	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN EuroHPC/CINECA Italy	1,824,768	241.20	306.31	7,494
8	**MareNostrum 5 ACC** - BullSequana XH3000, Xeon Platinum 8460Y+ 32C 2.3GHz, NVIDIA H100 64GB, Infiniband NDR, EVIDEN EuroHPC/BSC Spain	663,040	175.30	249.44	4,159
9	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
10	**Eos NVIDIA DGX SuperPOD** - NVIDIA DGX H100, Xeon Platinum 8480C 56C 3.8GHz, NVIDIA H100, Infiniband NDR400, Nvidia NVIDIA Corporation United States	485,888	121.40	188.65	

Ref: Top 500 list released on June 1st, 2024

GPU APIs



- low-level: memory management, CPU-GPU data transfer, and scheduling and execution of parallel processing tasks on GPU
- high-level: modules and libraries optimized for specific HPC workloads
- debugging/profiling tools to optimize codes

Cross-platform APIs: **DirectCompute**, **OpenCL**, **SYCL**, ...

GPU Programming. Why. When. How (Nov. 12-14)

GPU programming models

- directive-based models
- non-portable kernel-based models
- portable kernel-based models
- high-level programming languages

Directive-based models

The serial code is annotated with directives telling compilers to run specific loops and regions on GPU.

Two representative directives-based programming models are **OpenACC** and **OpenMP**.

```c
#include <stdio.h>
#include <openacc.h>

#define NX 102400

int main(void)
{
    double vecA[NX], vecB[NX], vecC[NX];
    int i;

    /* Initialization of the vectors */
    for (i = 0; i < NX; i++) {
        vecA[i] = 1.0;
        vecB[i] = 2.0;
    }

    #pragma acc kernels
    for (i = 0; i < NX; i++) {
        vecC[i] = vecA[i] + vecB[i];
    }

    return 0;
}
```

```fortran
program main
  implicit none

  integer, parameter :: nx = 102400
  integer :: i

  double precision :: vecA(nx), vecB(nx), vecC(nx)

  do i = 1, nx
     vecA(i) = 1.0
     vecB(i) = 1.0
  end do

  !$omp target
  do i = 1, nx
    vecC(i) = vecA(i) + vecB(i)
  end do
  !$omp end target

end program
```

Non-portable kernel-based models

Developers write low-level codes that directly communicates with GPU and its hardware.

Two representative programming models are CUDA and HIP.

```
int main(int argc, const char * argv[])
{

    const int NX = 2048;
    int size_array = sizeof(double) * NX;

    double *h_vecA = (double *)malloc(size_array); // 'h' for host (CPU)
    double *h_vecB = (double *)malloc(size_array);
    double *h_vecC = (double *)malloc(size_array);

    // initialization for h_vecA and h_vecB

    double *d_vecA, *d_vecB, *d_vecC; // 'd' for device (GPU)
    cudaMalloc((void **)&d_vecA, size_array);
    cudaMalloc((void **)&d_vecB, size_array);
    cudaMalloc((void **)&d_vecC, size_array);
    cudaMemcpy(d_vecA, h_vecA, size_array, cudaMemcpyHostToDevice);
    cudaMemcpy(d_vecB, h_vecB, size_array, cudaMemcpyHostToDevice); // copy data from CPU to GPU

    const int block_size = 128;
    int grid_size = (NX + block_size - 1) / block_size;
    array_addition<<<grid_size, block_size>>>(d_vecA, d_vecB, d_vecC);

    cudaMemcpy(h_vecC, d_vecC, size_array, cudaMemcpyDeviceToHost); // fetch data from GPU) to CPU

    free(h_vecA);
    free(h_vecB);
    free(h_vecC);
    cudaFree(d_vecA);
    cudaFree(d_vecB);
    cudaFree(d_vecC);

    return 0;
}
```

Portable kernel-based models

- Cross-platform portability ecosystems typically provide a higher-level abstraction layer which provide a convenient and portable programming model for GPU programming.

- For C++, the most notable cross-platform portability ecosystems are **Alpaka**, **Kokkos**, **OpenCL**, and **SYCL**.

- Pros and cons of cross-platform portability ecosystems
  - Pros
    - The amount of code duplication is minimized
    - Less knowledge of the underlying architecture is needed for initial porting
  - Cons
    - These models are relatively new and not very popular yet
    - Limited learning resources compared to CUDA
    - some low-level APIs are less user friendly

High-level programming languages