



alpaka Parallel Programming Library In a Nutshell

alpaka - Abstraction Library for Parallel Kernel Acceleration

Alpaka is...

- A parallel programming library: Accelerate your code by exploiting your hardware's parallelism!
- An abstraction library independent of hardware ecosystem: Create portable code that runs on CPUs and GPUs!
- Free & open-source software

The logo for alpaka, featuring the word "alpaka" in a blue sans-serif font. The letter "p" is stylized with an orange outline of a alpaca's head and neck.

Problem of HPC Systems?

Heterogenous Hardware Ecosystem!

TOP500

- 1 Frontier(USA) 1.194 Exaflop/s, AMD EPYC CPU + AMD Instinct GPU
- 2 Aurora(USA) 585 Petaflop/s, Intel Xeon CPU + Intel GPU Max
- 3 Eagle(USA) 561 Petaflop/s, Intel Xeon CPU + NVIDIA H100 GPU
- 4 Fugaku (Japan) 442 Petaflop/s, Fujitsu A64FX CPU
- 5 Lumi (Finland) 380 Petaflop/s, AMD EPYC CPU + AMD Instinct GPU
- 6 Leonardo (Italy) 238.7 Petaflop/s, Intel Xeon CPU + NVIDIA A100 GPU
- 7 MareNostrum (Spain) 183.2 Petaflops/s, Intel Xeon CPU + NVIDIA H100 GPU
- 8 Summit(USA) 148.3 Petaflops/s, IBM Power9 CPU + NVIDIA Volta GPU
- 9 Eos(USA) 121.4 Petaflops/s, Intel Xeon CPU + NVIDIA H100 GPU
- 10 Sierra(USA) 94 Petaflops/s, IBM POWER9 CPU + NVIDIA Tesla V100 GPU

THE LIST

www.top500.org

11/2023 Highlights

The 62nd edition of the TOP500 shows five new or upgraded entries in the top 10 but the Frontier system still remains the only true exascale machine with an HPL score of 1.194 Exaflop/s.

The Frontier system at the Oak Ridge National Laboratory, Tennessee, USA remains the No. 1 system on the TOP500 and is still the only system reported with an HPL performance exceeding one Exaflop/s. Frontier brought the pole position back to the USA one year ago on the June 2022 listing and has since been remeasured with an HPL score of 1.194 Exaflop/s.

Frontier is based on the latest HPE Cray EX235a architecture and is equipped with AMD EPYC 64C 2GHz processors. The system has 8,699,904 total cores, a power efficiency rating of 52.59 gigaflops/watt, and relies on HPE's Slingshot 11 network for data transfer.

The Aurora system at the Argonne Leadership Computing Facility, Illinois, USA is currently being commissioned and will at full scale exceed Frontier with a peak performance of 2 Exaflop/s. It was submitted with a measurement on half of the final system achieving 585 Petaflop/s on the HPL benchmark which secured the No. 2 spot on the TOP500.

Aurora is built by Intel based on the HPE Cray EX - Intel Exascale Compute Blade which uses Intel Xeon CPU Max Series processors and Intel Data Center GPU Max Series accelerators which communicate through HPE's Slingshot-11 network interconnect.

The Eagle system installed in the Microsoft Azure cloud in the USA is newly listed as No. 3. This Microsoft NDv5 system is based on Intel Xeon Platinum 8480C processors and NVIDIA H100 accelerators and achieved an HPL score of 561 Pflop/s.

[read more »](#)

List Statistics

Vendors System Share

1 **Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE

2 **Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel

3 **Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft

4 **Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu

5 **LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE

6 **Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN

7 **Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-

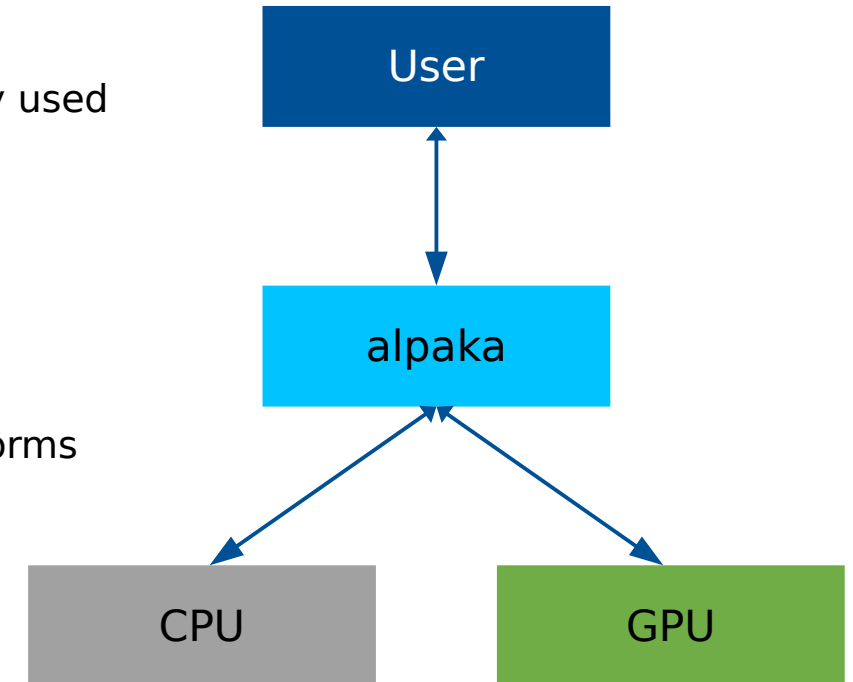
alpaka's purpose

Without alpaka

- Hardware ecosystem is heterogenous and multiple hardware types commonly used (CPUs, GPUs, ...)
- Platforms not inter-operable → parallel programs not easily portable

alpaka: one API to rule them all

- **Abstraction** (not hiding!) of the underlying hardware, compiler and OS platforms
- **Easy change of the backend**
 - Code needs only minor adjustments to support different accelerators
- **Easy indexing of threads in kernels**
- **Easy setup of the type of parallelism** (Block sizes in grid, Thread sizes in block...)
- **Heterogenous Programming**: Using different backends in a synchronized manner



Switching the Accelerator Easily

- alpaka provides a number of pre-defined Accelerators in the `acc` namespace.
- For GPUs:
 - `AccGpuCudaRt` for NVIDIA GPUs
 - `AccGpuHipRt` for AMD and NVIDIA GPUs
- For CPUs
 - `AccCpuFibers` based on Boost.fiber
 - `AccCpuOmp2Blocks` based on OpenMP 2.x
 - `AccCpuOmp4` based on OpenMP 4.x
 - `AccCpuTbbBlocks` based on TBB
 - `AccCpuThreads` based on `std::thread`

```
// Example: CUDA GPU accelerator
using Acc = alpaka::AccGpuCudaRt<Dim,
Idx>;

// Switch to AMD: HIP GPU accelerator
using Acc = alpaka::AccGpuHipRt<Dim,
Idx>;

// Switch to CPU Omp: CPU accelerator
using Acc =
alpaka::AccCpuOmp2Blocks<Dim, Idx>;
```

Programming with alpaka

- C++ only!
- Header-only library: No additional runtime dependency introduced
- Modern library: alpaka is written entirely in C++17
- Supports a wide range of modern C++ compilers (g++, clang++, Apple LLVM, MS Visual Studio)
- Portable across operating systems: Linux, macOS, Windows are supported
- Alpaka directly uses vendor API's. For example: Alpaka cuda backend uses Cuda API directly etc.



alpaka in the wild - example use case

PICongPU: <https://github.com/ComputationalRadiationPhysics/picongpu>

- Fully relativistic, manycore, 3D3V particle-in-cell (PIC) code
- Central algorithm in plasma physics
- Scalable to more than 18,000 GPUs
- Developed at Helmholtz-Zentrum Dresden-Rossendorf
- 2020, only NVIDIA was running now AMD!!!



alpaka is free software (MPL 2.0). Find us on GitHub!

Our GitHub organization: <https://www.github.com/alpaka-group>

- Contains all alpaka-related projects, documentation, samples, ...
- New contributors welcome!

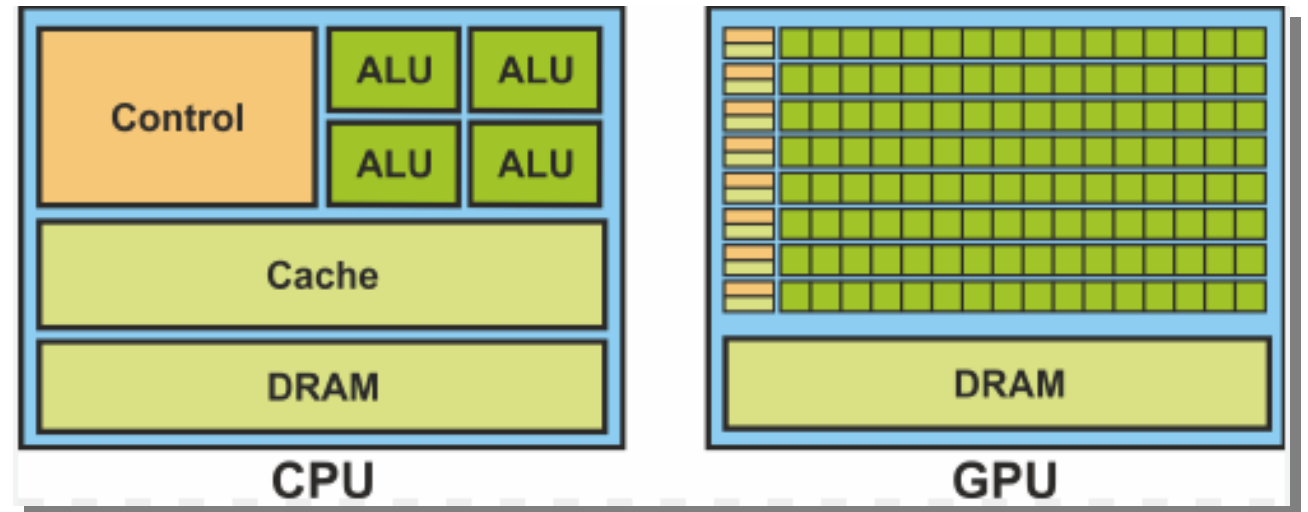
The library: <https://www.github.com/alpaka-group/alpaka>

- Full source code
- Issue tracker
- Installation instructions
- Small examples



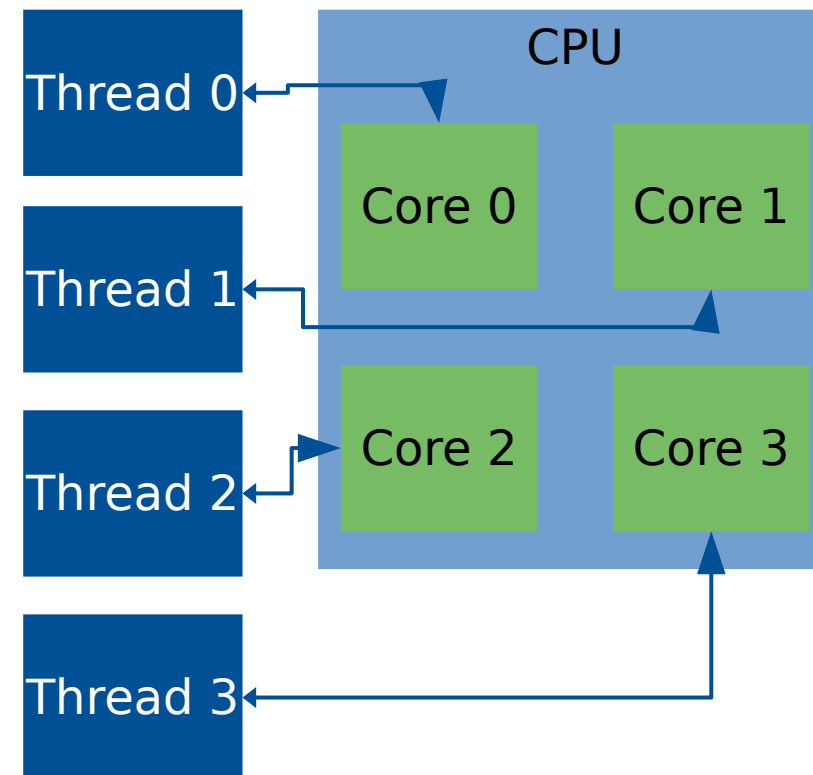
Main difference between CPU and GPU programming?

- CPU:
 - 1 thread per core and not many cores.
 - High cache.
 - Low memory bandwidth(BW).
- GPU:
 - Massively parallel many threads per core (Streaming Multiprocessor) and many cores.
 - Low cache.
 - High memory BW.



Basics: Thread mapping on CPUs

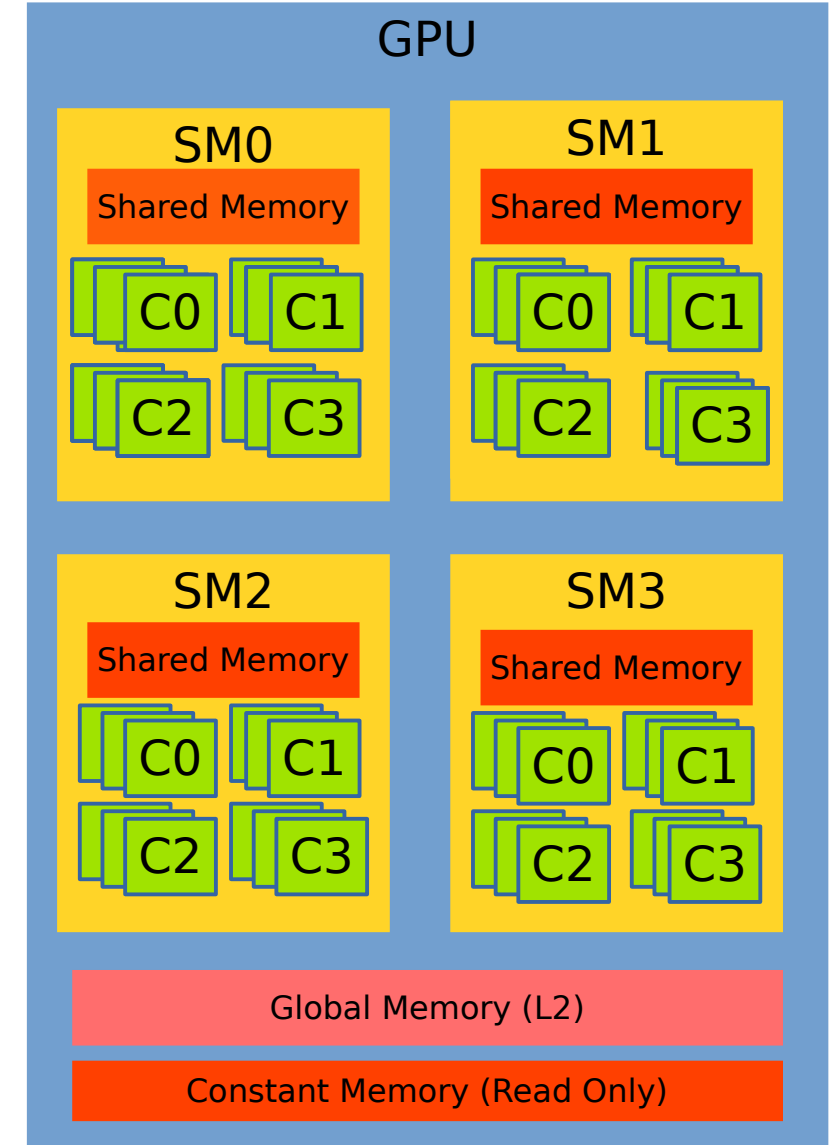
- CPU consists of multiple cores
 - Because of simultaneous multithreading there can be more logical than physical cores!
- alpaka Threads are executed by CPU cores.
- Single thread per block. Single block per core.
- Multiple elements per thread.



Thread Mapping on GPUs

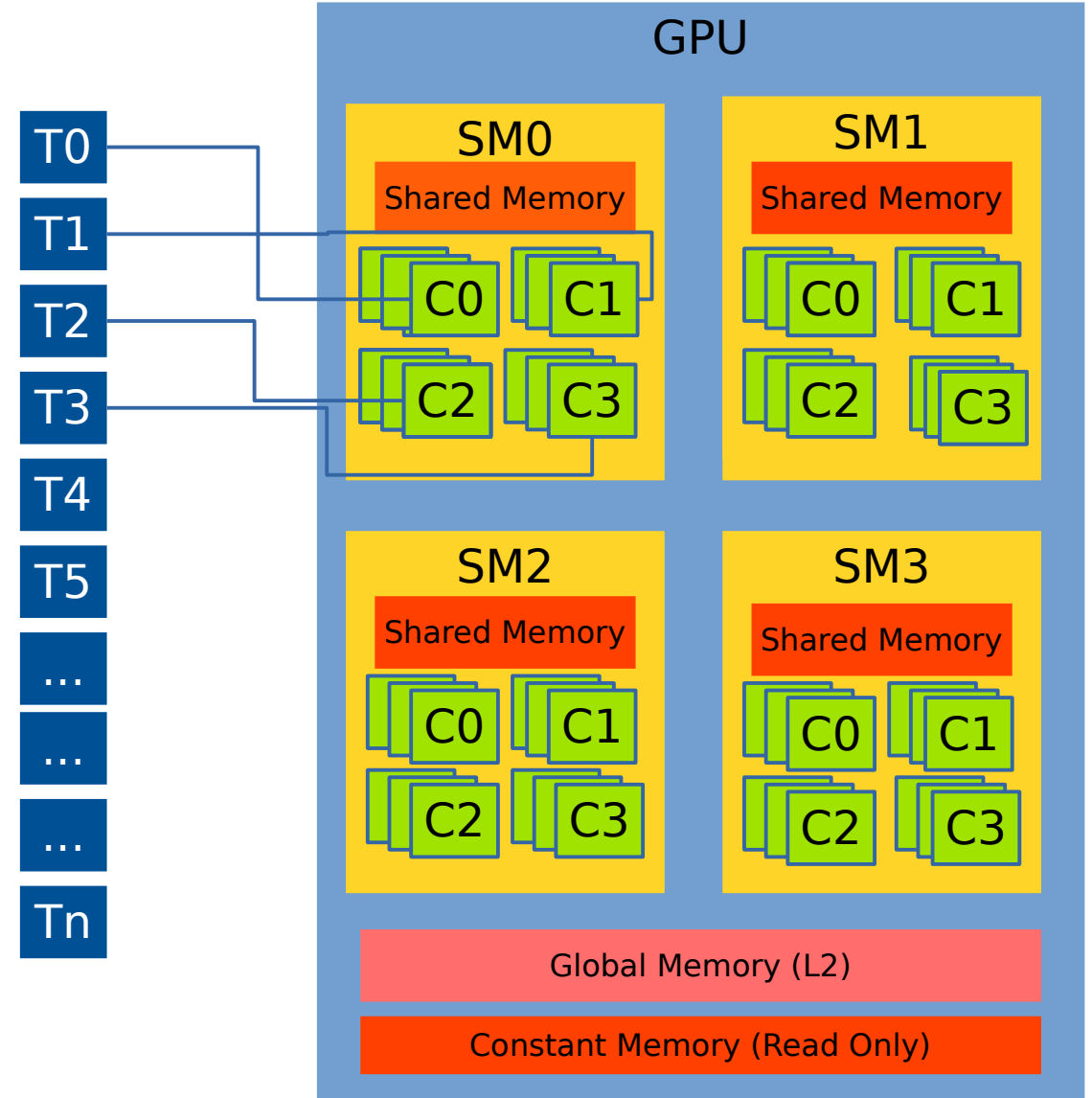
- GPU consists of **streaming multiprocessors** (SMs)
- Each SM consists of multiple **cores**
- A block has many threads and threads in a block are executed synchronously in the same SM. (SM can run more than one block)
- Each thread is matched to a core.

T0
T1
T2
T3
T4
T5
...
...
...
Tn



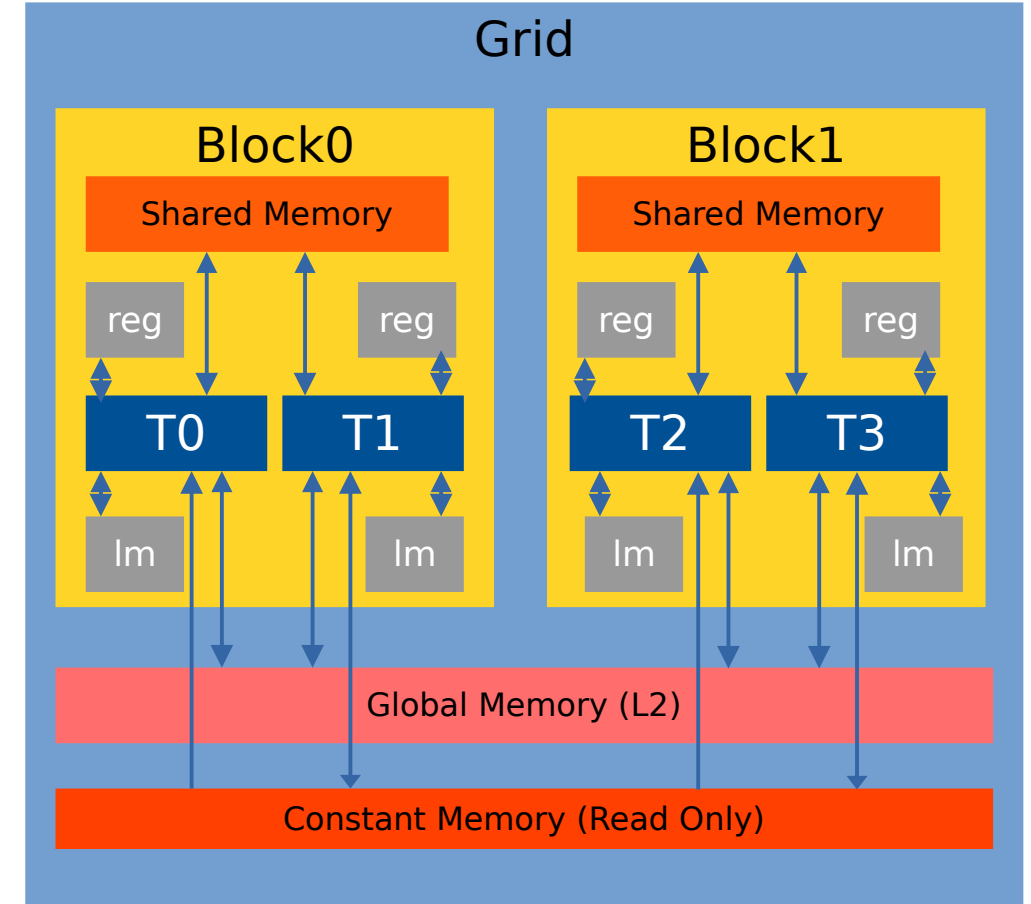
How distribute threads between SMs while matching each thread to a core?

- Main determinants of mapping threads to the SMs:
 - Number of cores SM,
 - Register size in each SM,
 - Shared memory usage of each thread,
 - Threads per SM, Threads per Block, Blocks per Device
- Memory latencies: Global Memory and Constant memory has different latencies.
- Memory sizes: Size of shared memory used by threads in a block or blocks assigned to an SM



Alpaka proposes and validates WorkDivision!

- WorkDivision data structure consists:
 - Number of Blocks per grid
 - Number of Threads per block
 - Elements per thread
- Alpaka validates and proposes correct parallelisation strategies to map threads to SMs



Easy definition, validation of Type of Parallelism by WorkDivision

- Determines the number of kernel instantiations
- Determines the type of parallelism
 - Dimensions of a grid in terms of blocks,
 - Dimensions of a block in terms of threads
 - Elements per thread

```
// Define the work division
// The workdiv is divided in three levels of parallelization:
// - grid-blocks:      The number of blocks in the grid
// - block-threads:    The number of threads per block (parallel, synchronizable).
// - thread-elements:  The number of elements per thread (sequential, not synchronizable)
//                      Each kernel has to execute its elements sequentially.

using Vec = alpaka::Vec<Dim, Idx>;
auto const elementsPerThread = Vec::all(static_cast<Idx>(1));
auto const threadsPerGrid = Vec{4, 2, 4};
using WorkDiv = alpaka::WorkDivMembers<Dim, Idx>;
WorkDiv const workDiv = alpaka::getValidWorkDiv<Acc>( devAcc, threadsPerGrid,
|             elementsPerThread, false, alpaka::GridBlockExtentSubDivRestrictions::Unrestricted);

// Instantiate the kernel function object
HelloWorldKernel helloWorldKernel;

// Run the kernel
// To execute the kernel, you have to provide the
// work division as well as the additional kernel function parameters.
alpaka::exec<Acc>(queue, workDiv, helloWorldKernel/* put kernel arguments here */);
```

Alpaka Features [Todo:Explain shortly?]

- Buffer
- Memory Fences
- Queues
- Events
- Streams
- <https://alpaka.readthedocs.io/en/latest/advanced/rationale.html>

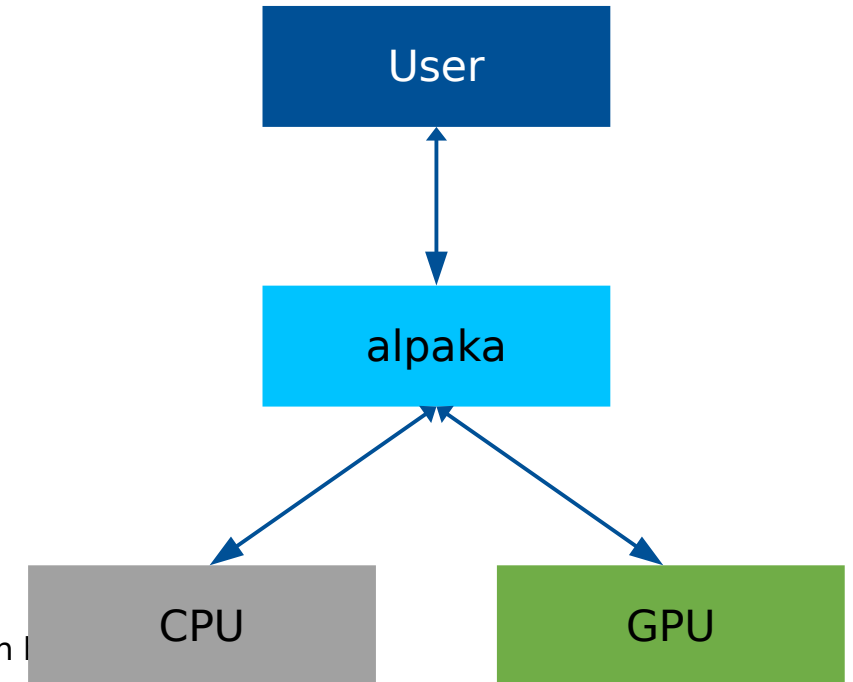
As a summary

Without alpaka

- Multiple hardware types commonly used (CPUs, GPUs, ...)
- Increasingly heterogeneous hardware configurations available
- Platforms not inter-operable → parallel programs not easily portable

alpaka: one API to rule them all

- **Abstraction** (not hiding!) of the underlying hardware & software platforms
 - AMD, Nvidia, Intel GPUs, CPU
- **Easy change of the backend**
 - Code needs only minor adjustments to support different accelerators
- **Easy indexing of threads in kernels**
- **Easy setup of the type of parallelism by WorkDivision** (Block sizes in grid, Thread sizes in l
- **Heterogenous Programming**: Using different backends in a synchronized manner



Thank you! If you use alpaka for your research, please cite one of the following publications:

Matthes A., Widera R., Zenker E., Worpitz B., Huebl A., Bussmann M. (2017): Tuning and Optimization for a Variety of Many-Core Architectures Without Changing a Single Line of Implementation Code Using the Alpaka Library. In: Kunkel J., Yokota R., Taufer M., Shalf J. (eds) High Performance Computing. ISC High Performance 2017. Lecture Notes in Computer Science, vol 10524. Springer, Cham, DOI: [10.1007/978-3-319-67630-2_36](https://doi.org/10.1007/978-3-319-67630-2_36).

E. Zenker et al., “Alpaka – An Abstraction Library for Parallel Kernel Acceleration”, 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, 2016, pp. 631 – 640, DOI: [10.1109/IPDPSW.2016.50](https://doi.org/10.1109/IPDPSW.2016.50).

Worpitz, B. (2015, September 28). Investigating performance portability of a highly scalable particle-in-cell simulation code on various multi-core architectures. Zenodo. DOI: [10.5281/zenodo.49768](https://doi.org/10.5281/zenodo.49768).