Alpaka – An Abstraction Library for Parallel Kernel Acceleration

Erik Zenker[1,2], Benjamin Worpitz[1,2], René Widera[1],Axel Huebl[1,2],
Guido Juckeland[1], Andreas Knüpfer[2], Wolfgang E. Nagel[2], Michael Bussmann[1]

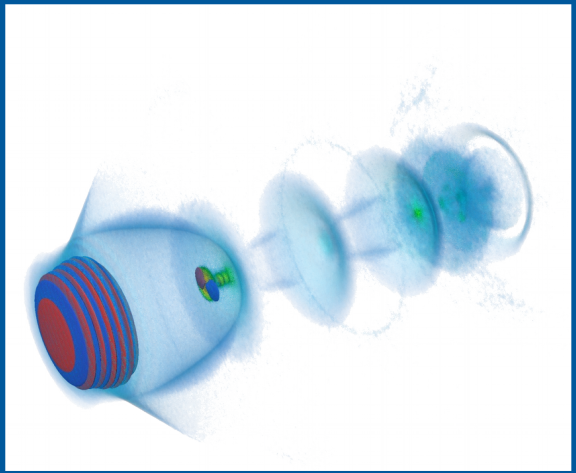
[1] Helmholtz-Zentrum Dresden – Rossendorf
[2] Technische Universität Dresden

We Have Lasers : Risk of Forest Fires !

PIConGPU

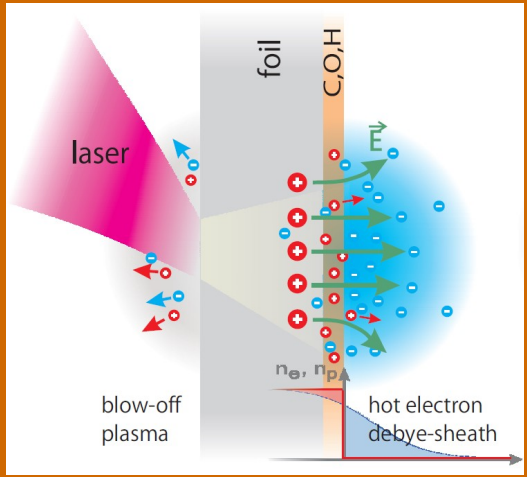
Electron Acceleration with Lasers

- Compact X-Ray sources



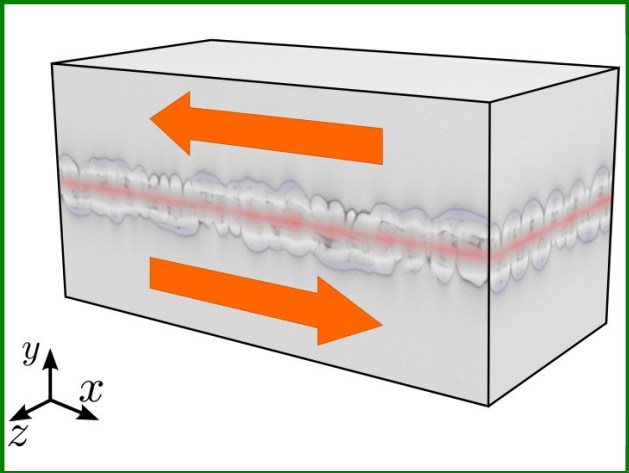
Ion Acceleration with Lasers

- Tumor Therapy



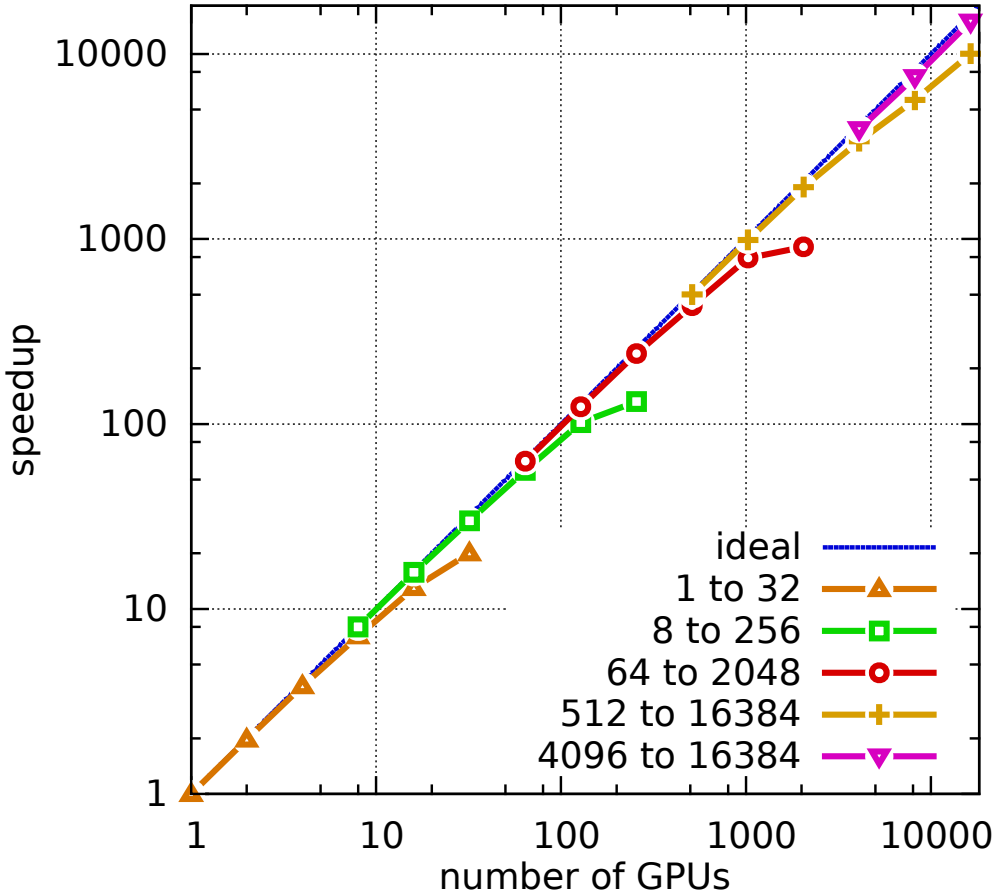
Plasma Instabilities

- Astrophysics

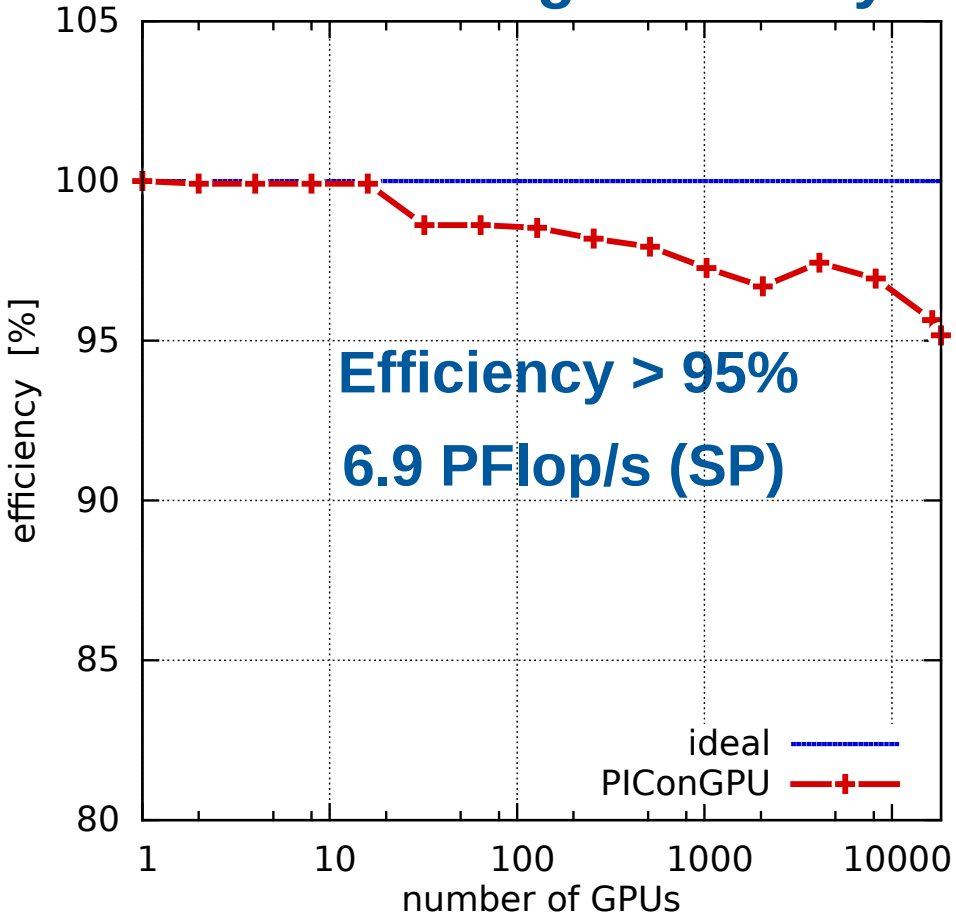


PIConGPU – Scales up to **18,432** GPUs

strong scaling

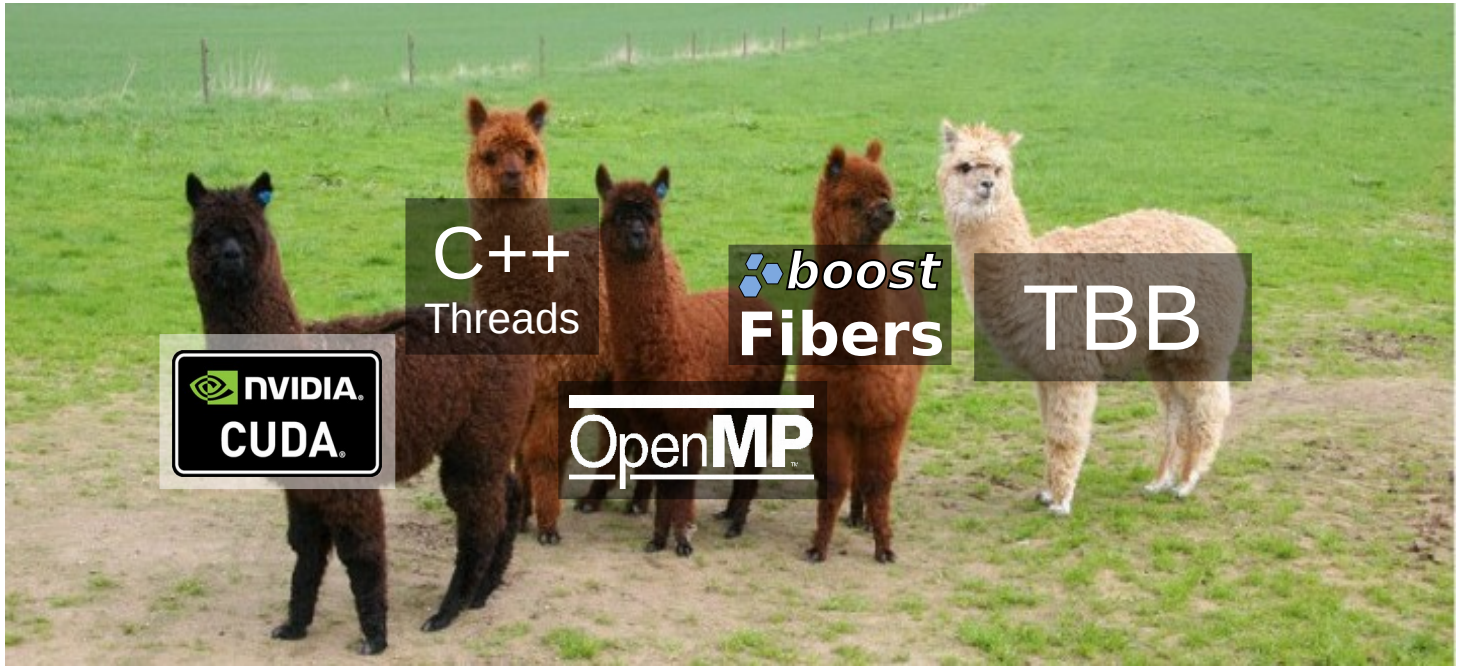


weak scaling efficiency



Alpaka

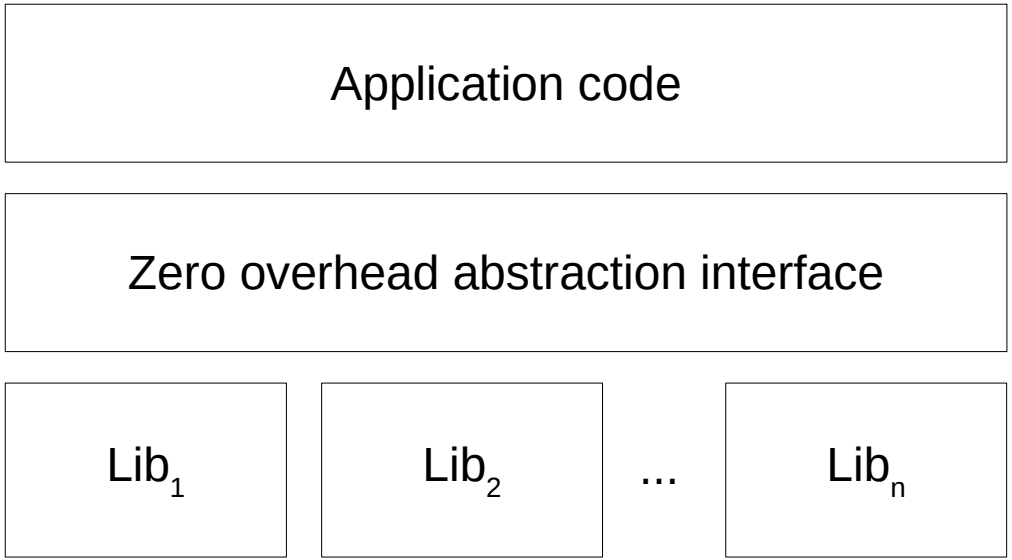
Good News: There are **Alpaka**s on the Compute Meadow



- Single zero overhead interface to existing parallelism models
- Single source C++11 kernels
- Data-agnostic memory model

A Uniform Interface to All These Programming Models ?

- Interface is defined by a set of free functions with template arguments
- Template arguments need to fulfill type requirements (concepts)
- Interface is extendable through more concepts implementations (models)



C++ compilers are able to almost completely remove abstraction layers

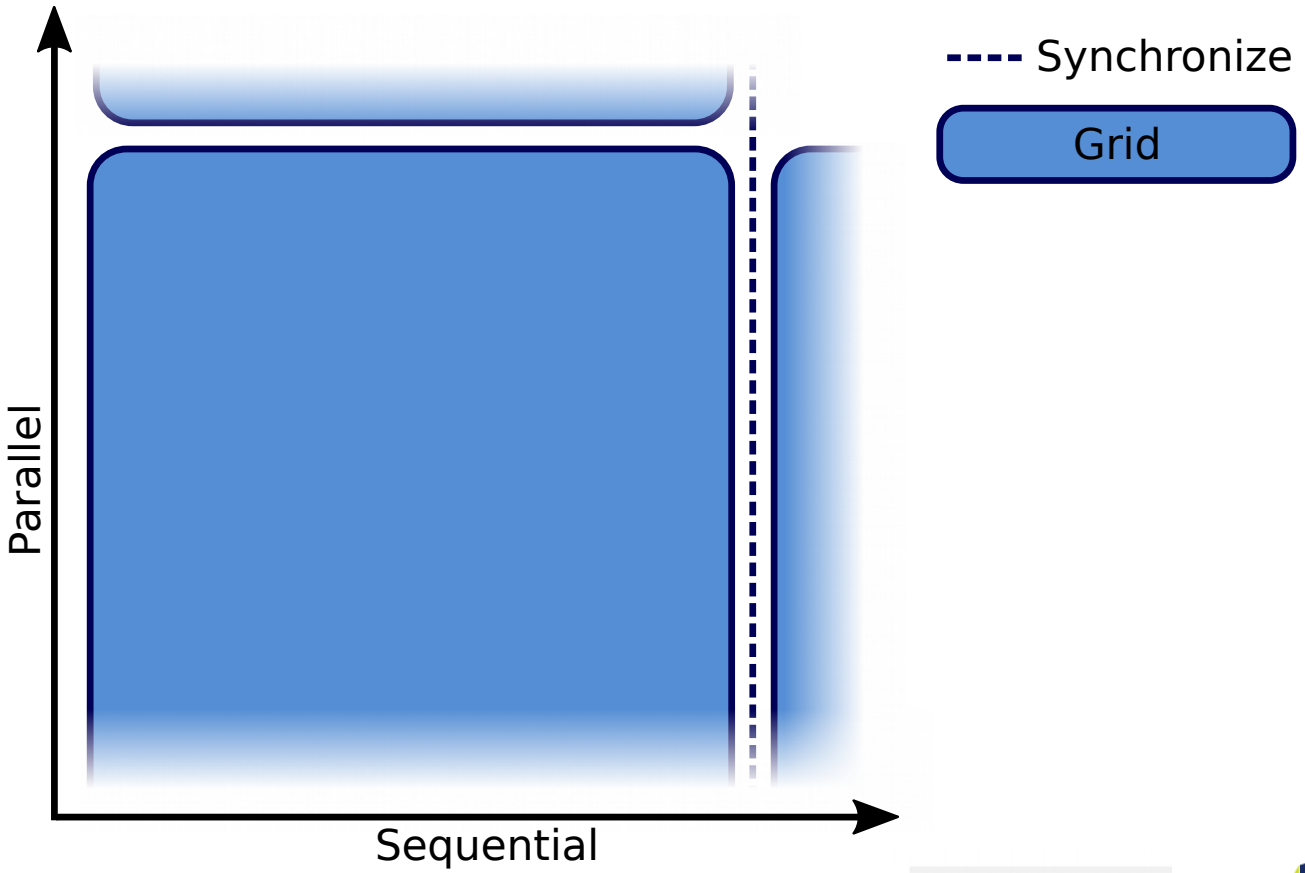
Heterogeneous Codes Need to be Maintainable

Heterogeneity	Testability	Sustainability	Optimizability	Openness
Write once, execute everywhere	Validate once, get correct results everywhere	Porting implies minimal code changes	Tune for good performance at minimum coding effort	Open source and open standards

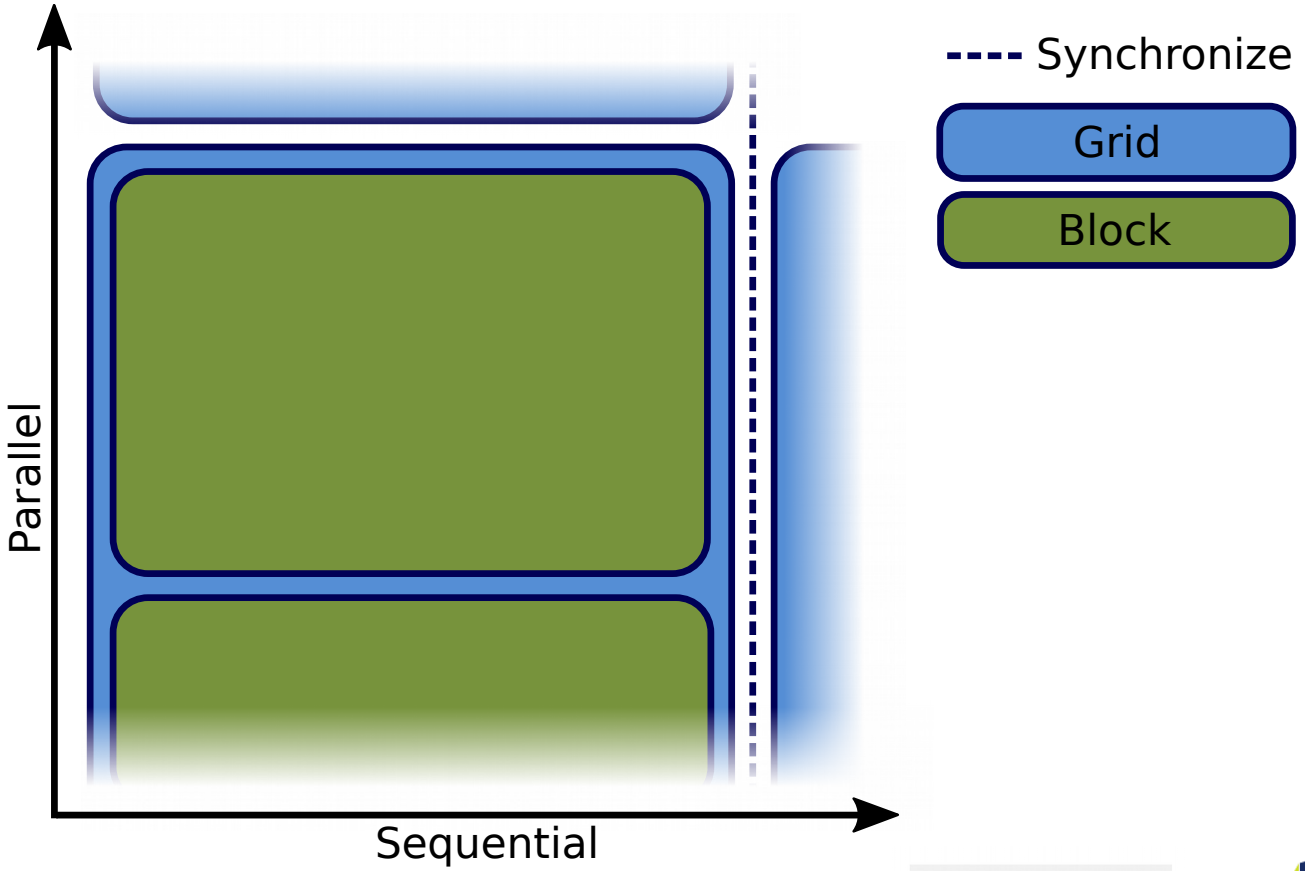
Single Source

René Widera, Erik Zenker, Guido Juckeland · Computational Radiation Physics · www.hzdr.de/crp
{ r.widera, e.zenker, g.juckeland }@hzdr.de

Abstract Hierarchical Redundant Parallelism Model

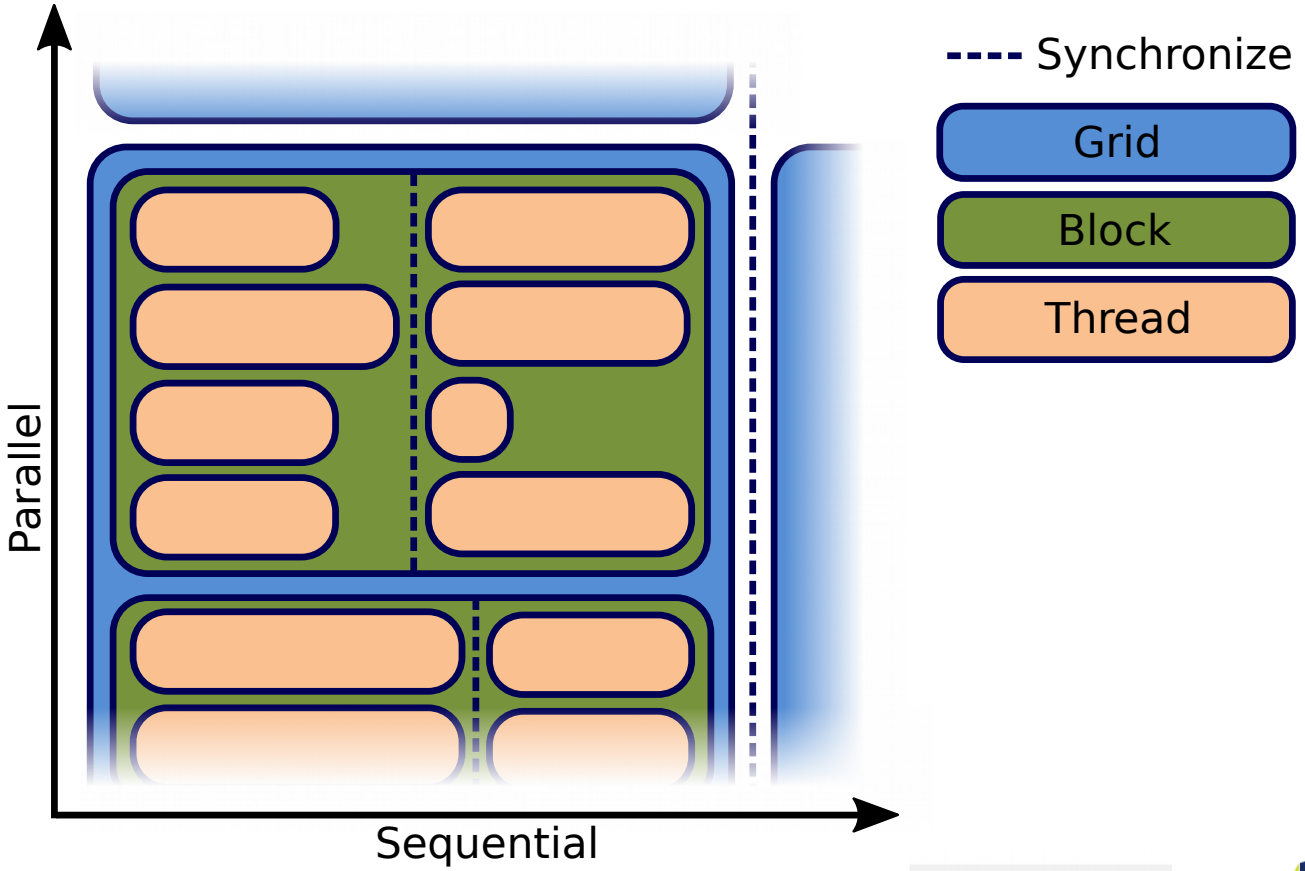


Abstract Hierarchical Redundant Parallelism Model

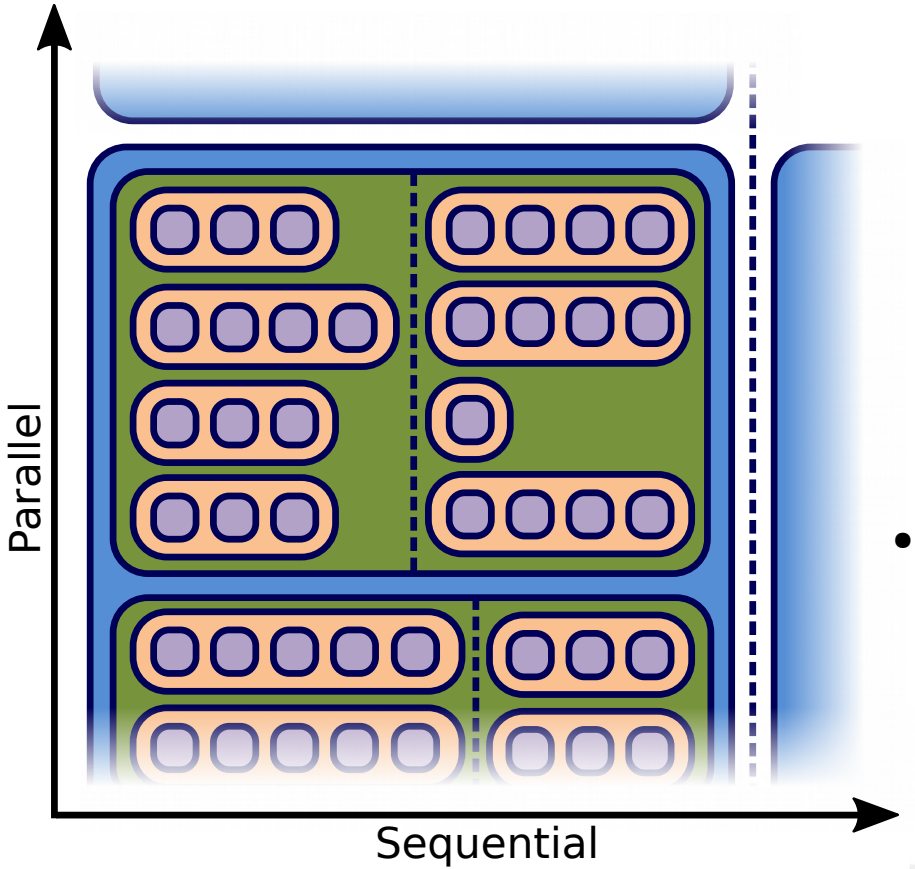


René Widera, Erik Zenker, Guido Juckeland · Computational Radiation Physics · www.hzdr.de/crp
{ r.widera, e.zenker, g.juckeland }@hzdr.de

Abstract Hierarchical Redundant Parallelism Model



Abstract Hierarchical Redundant Parallelism Model



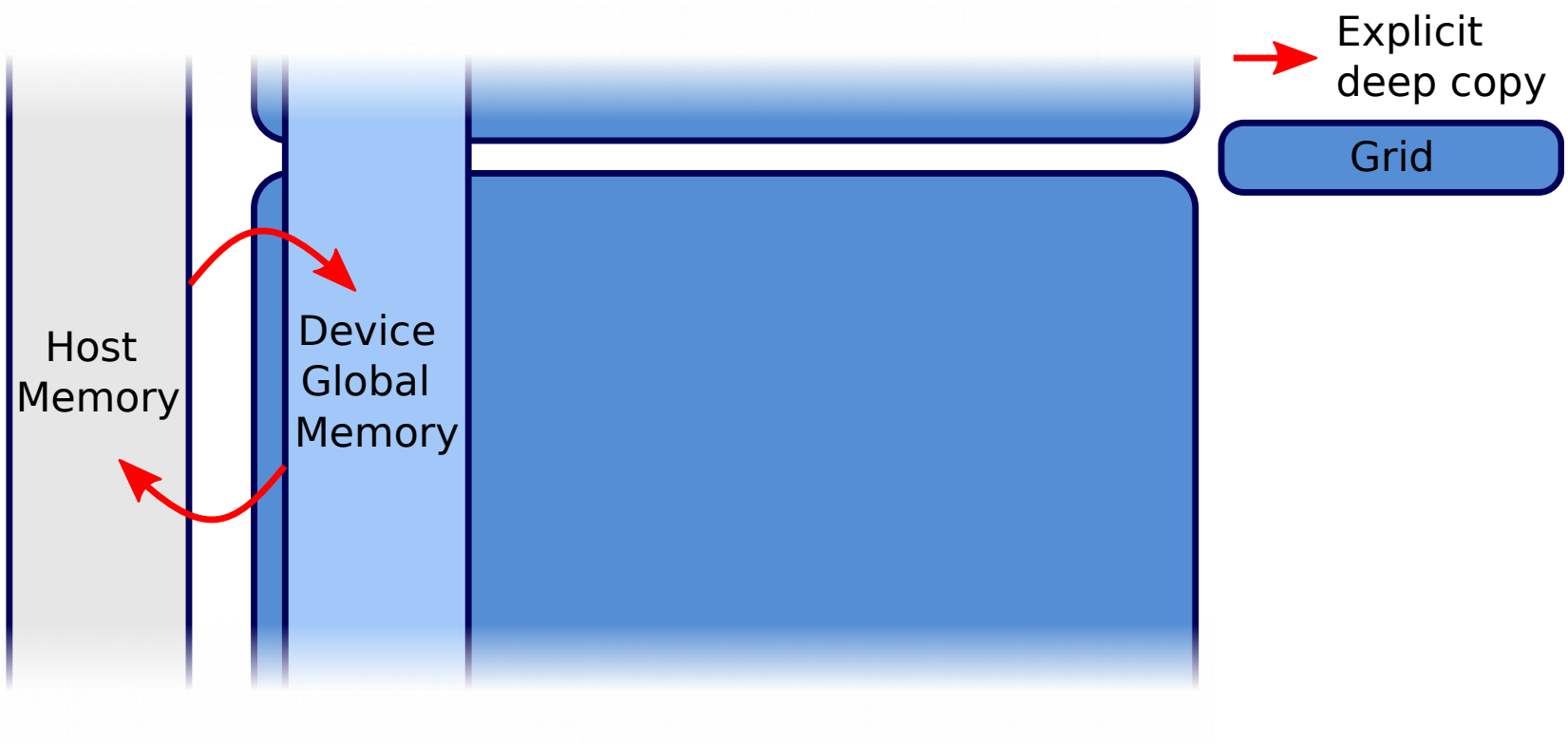
---- Synchronize

- Grid
- Block
- Thread
- Element

- Element level is an explicit sequential layer

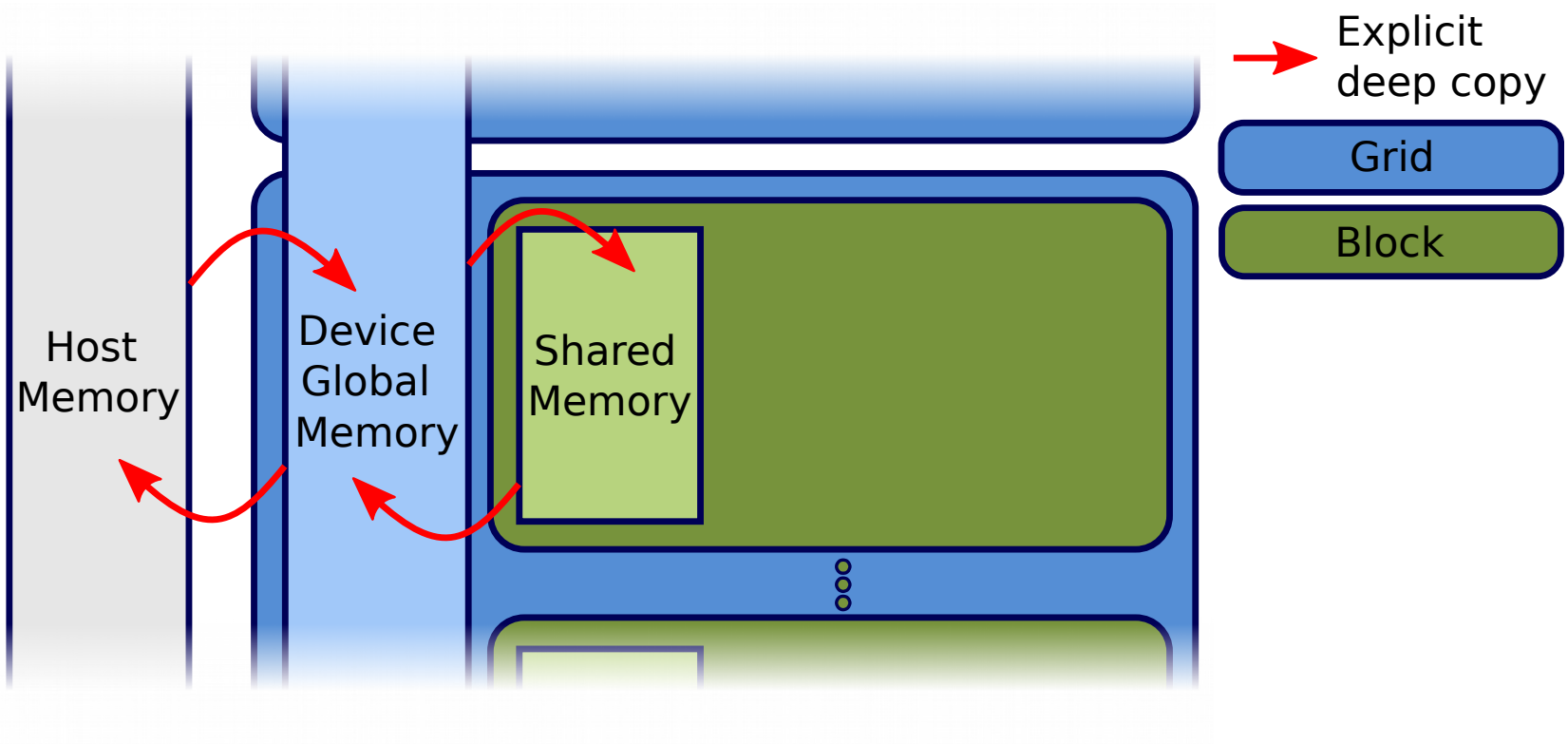
René Widera, Erik Zenker, Guido Juckeland · Computational Radiation Physics · www.hzdr.de/crp
{ r.widera, e.zenker, g.juckeland }@hzdr.de

Data Structure Agnostic Memory Model

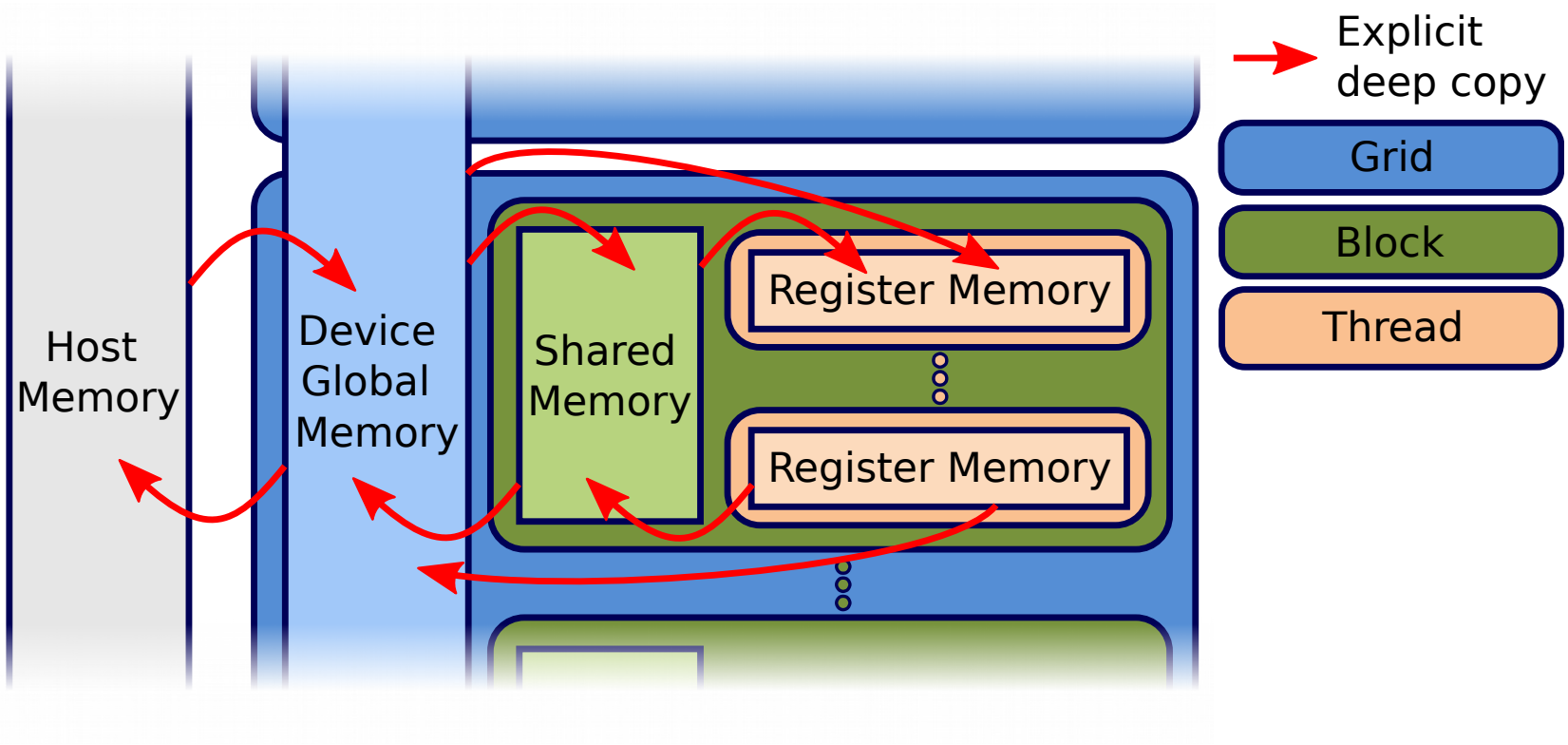


René Widera, Erik Zenker, Guido Juckeland · Computational Radiation Physics · www.hzdr.de/crp
{ r.widera, e.zenker, g.juckeland }@hzdr.de

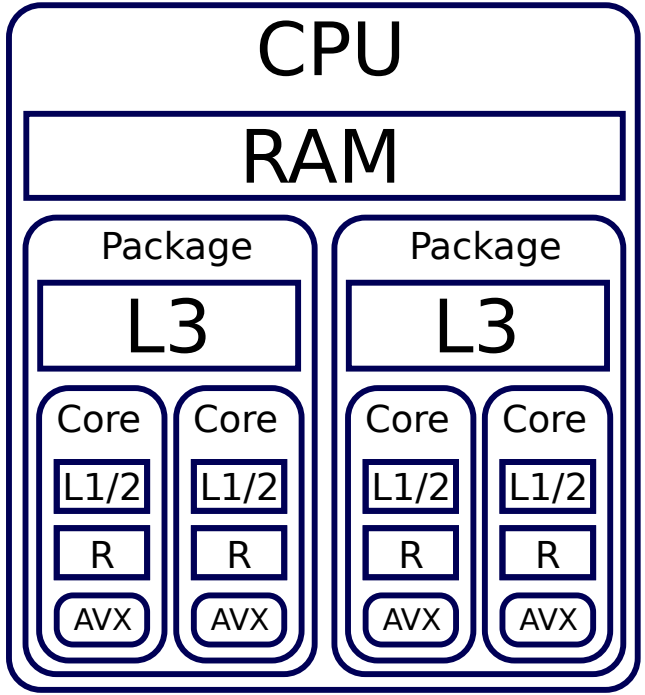
Data Structure Agnostic Memory Model



Data Structure Agnostic Memory Model

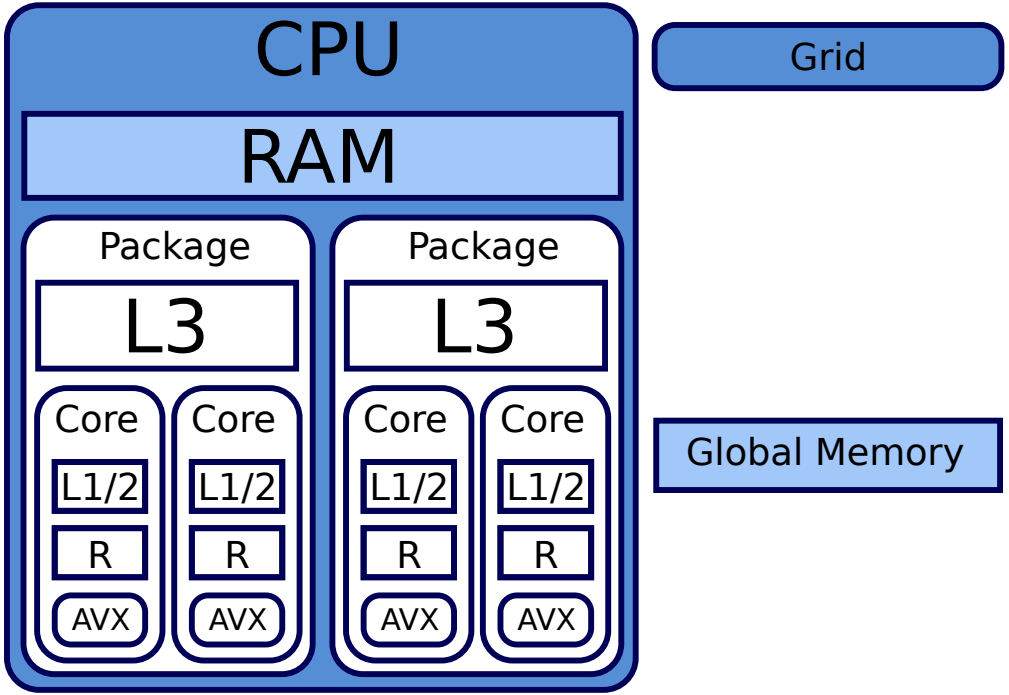


Map the Abstraction Model to your Desired Acceleration Back-End



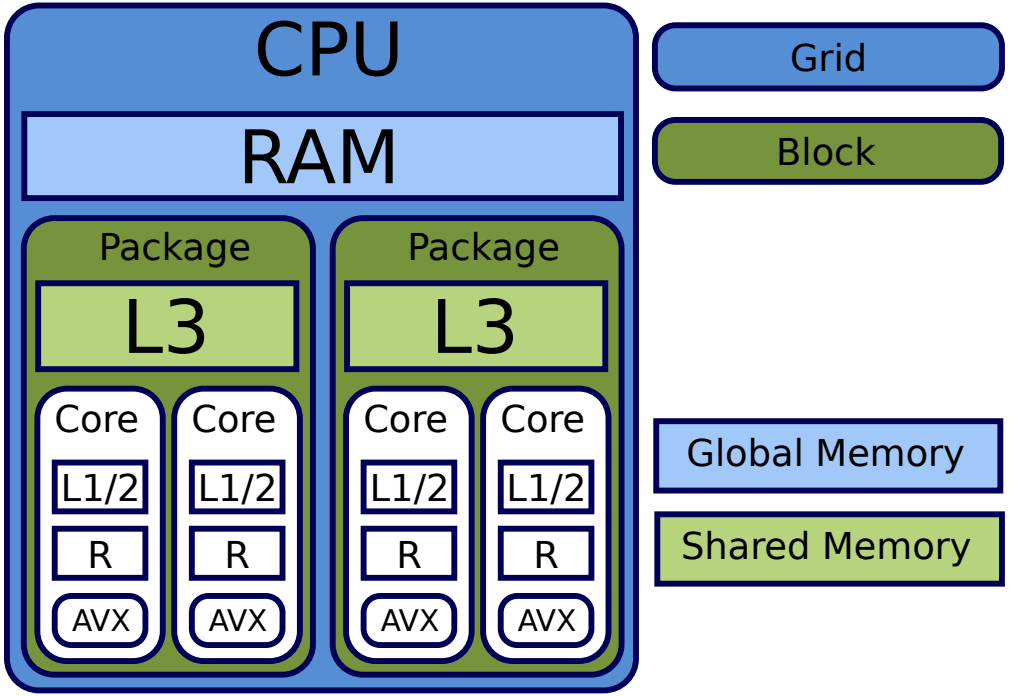
- Explicit mapping of parallelization levels to hardware

Map the Abstraction Model to your Desired Acceleration Back-End



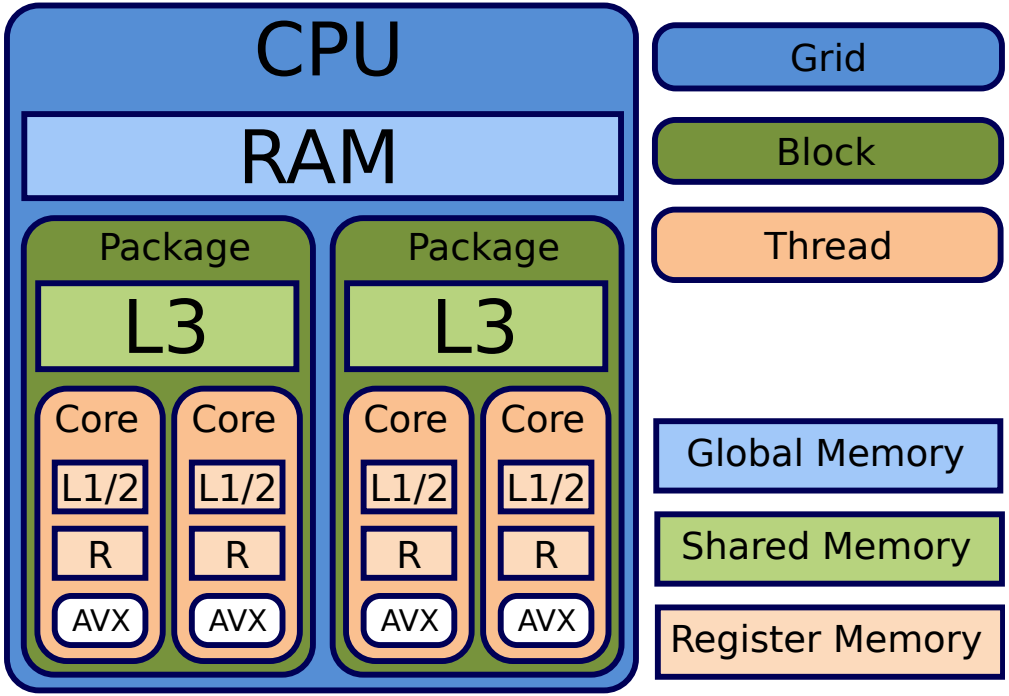
- Explicit mapping of parallelization levels to hardware

Map the Abstraction Model to your Desired Acceleration Back-End



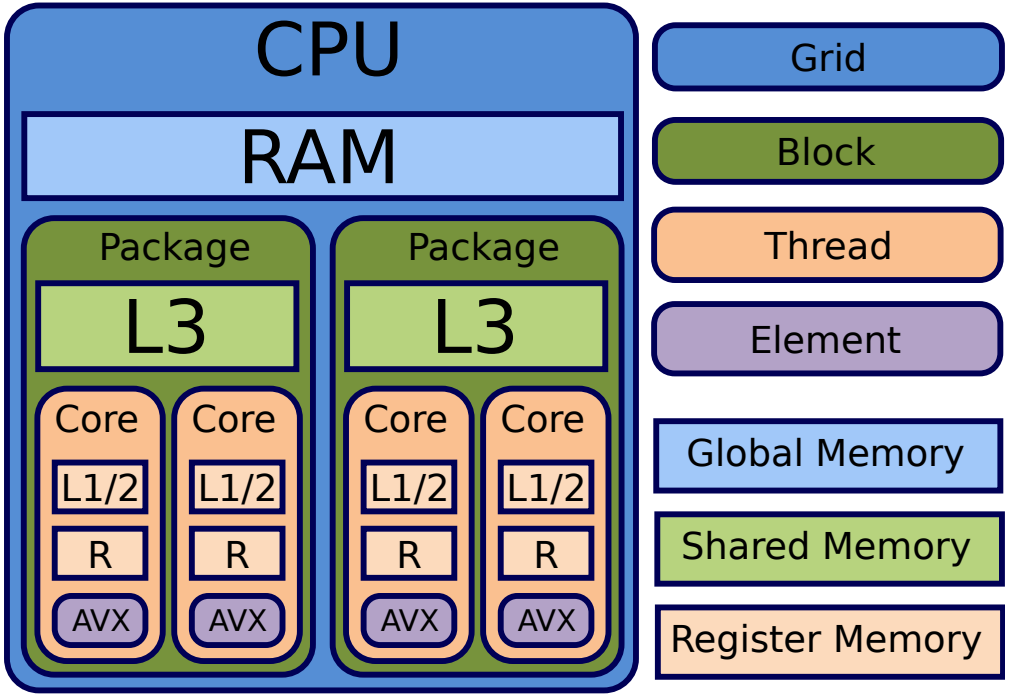
- Explicit mapping of parallelization levels to hardware

Map the Abstraction Model to your Desired Acceleration Back-End



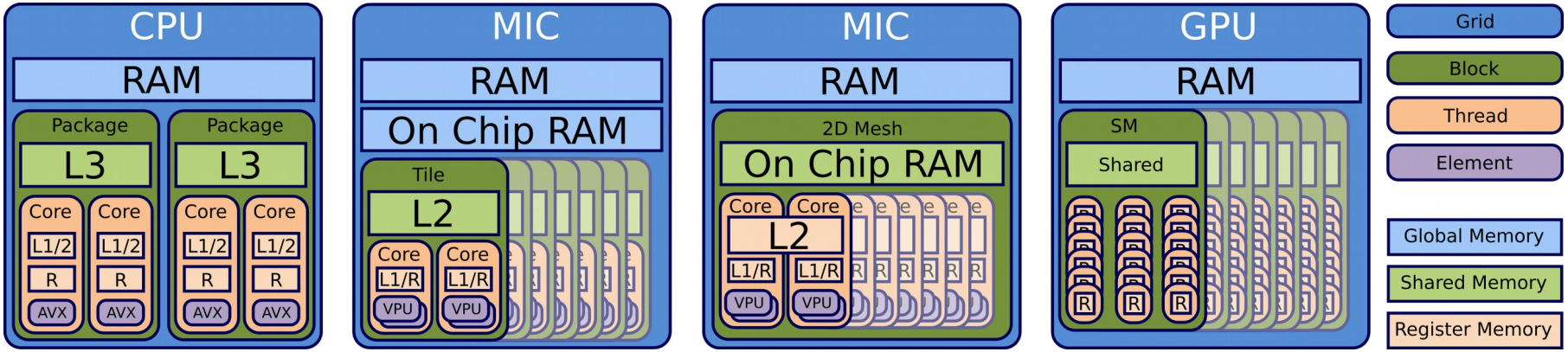
- Explicit mapping of parallelization levels to hardware

Map the Abstraction Model to your Desired Acceleration Back-End



- Explicit mapping of parallelization levels to hardware

Map the Abstraction Model to your Desired Acceleration Back-End



- Specific unsupported levels of the model can be ignored
- Abstract interface allows to extend the set of mappings

Alpaka : Vector Addition Kernel

```
struct VectorAdd
{
   template<typename TAcc, typename TElem, typename TSize>
   ALPAKA_FN_ACC auto operator()(
      TAcc const & acc,
      TSize const & numElements,
      TElem const * const X,
      TElem * const Y) const -> void {

      using alp = alpaka;
      auto globalIdx      = alp::idx::getIdx<alp::Grid, alp::Threads>(acc)[0u];
      auto elemsPerThread = alp::workdiv::getWorkDiv<alp::Thread, alp::Elems>(acc)[0u];

      auto begin = globalIdx * elemsPerThread;
      auto end   = min(begin + elemsPerThread, numElements);

      for(TSize i = begin; i < end; ++i){
         Y[i] = X[i] + Y[i];
      }

    }

};
```


René Widera, Erik Zenker, Guido Juckeland · Computational Radiation Physics · www.hzdr.de/crp
{ r.widera, e.zenker, g.juckeland }@hzdr.de

Alpaka : Initialization

```
// Configure Alpaka
using Dim     = alpaka::dim::DimInt<3u>
using Size    = std::size_t
using Acc     = alpaka::acc::AccCpuSerial<Dim, Size>;
using Host    = alpaka::acc::AccCpuSerial<Dim, Size>;
using Stream  = alpaka::stream::StreamCpuSync;
using WorkDiv = alpaka::workdiv::WorkDivMembers<Dim, Size>;
using Elem    = float;


// Retrieve devices and stream
DevHost devHost ( alpaka::dev::DevMan<Host>::getDevByIdx(0) );
DevAcc  devAcc  ( alpaka::dev::DevMan<Acc>::getDevByIdx(0) );
Stream  stream  ( devAcc);

// Specify work division
auto elementsPerThread ( alpaka::Vec<Dim, Size>::ones() );
auto threadsPerBlock   ( alpaka::Vec<Dim, Size>::all(2u) );
auto blocksPerGrid     ( alpaka::Vec<Dim, Size>(4u, 8u, 16u) );

WorkDiv workdiv(alpaka::workdiv::WorkDivMembers<Dim, Size>(blocksPerGrid,
                                                           threadsPerBlock,
                                                           elementsPerThread));
```

Alpaka : Call the Kernel

```
// Memory allocation and host to device memory copy
auto X_h = alpaka::mem::buf::alloc<int, int>(devHost, extent);
auto Y_h = alpaka::mem::buf::alloc<int, int>(devHost, extent);
auto X_d = alpaka::mem::buf::alloc<Val, Size>(devAcc, extent);
auto Y_d = alpaka::mem::buf::alloc<Val, Size>(devAcc, extent);

alpaka::mem::view::copy(stream, X_d, X_h, extent);
alpaka::mem::view::copy(stream, Y_d, Y_h, extent);


// Kernel creation and execution
VectorAdd kernel;
auto const exec(alpaka::exec::create<Acc>(
    workDiv,
    kernel,
    numElements
    alpaka::mem::view::getPtrNative(X_d),
    alpaka::mem::view::getPtrNative(Y_d)));

alpaka::stream::enqueue(stream, exec);


// Copy memory back to host
alpaka::mem::view::copy(stream, Y_h, Y_d, extent);
```

Compile to Almost Same PTX Code (DAXPY)

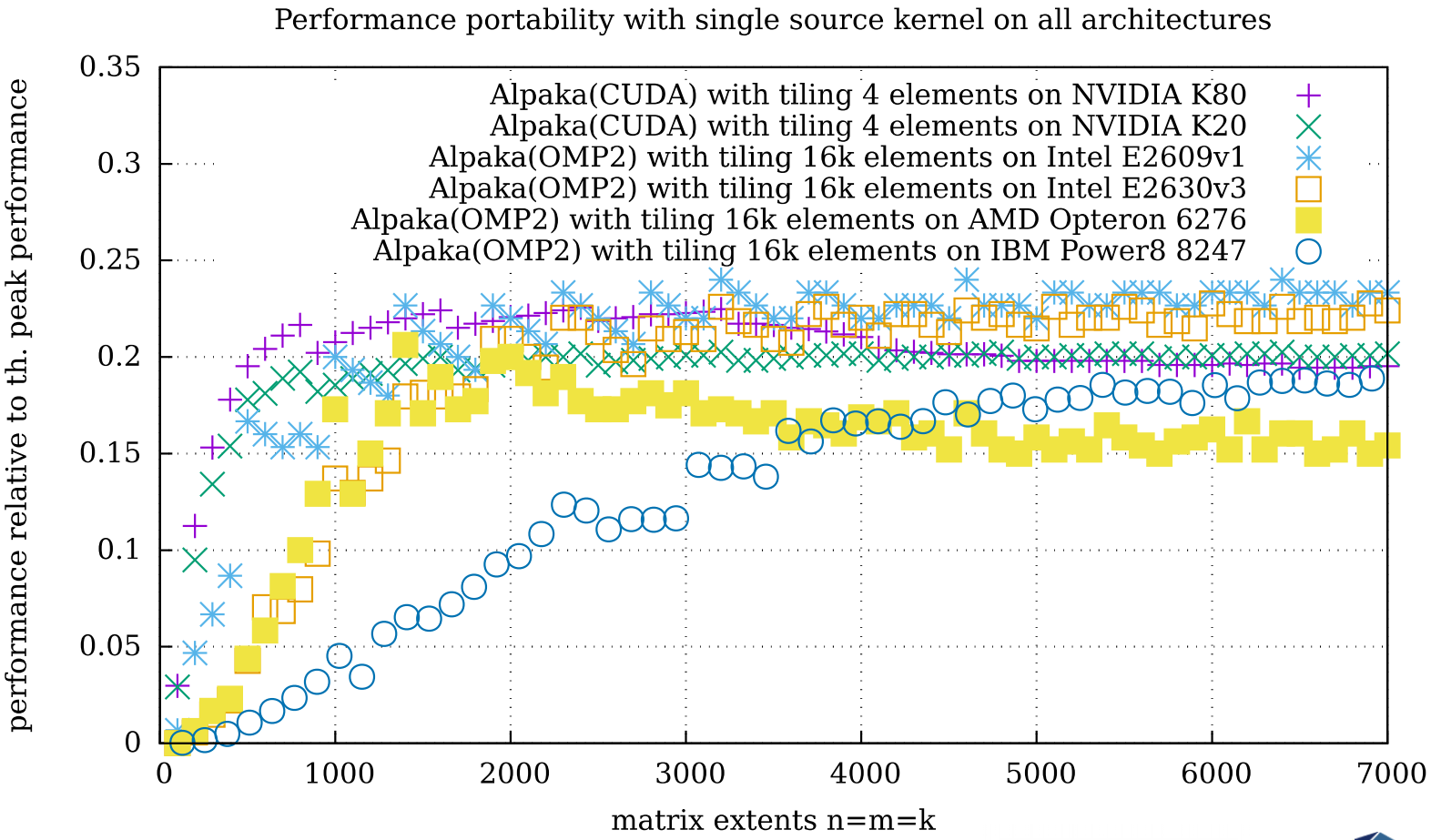
Alpaka CUDA PTX

```
mov.u32      %r3, %ctaid.x;
mov.u32      %r4, %ntid.x;
mov.u32      %r5, %tid.x;
mad.lo.s32   %r1, %r4, %r3, %r5;
setp.ge.s32 %p1, %r1, %r2;
@%p1 bra  BB6_2;

cvta.to.global.u64  %rd3, %rd2;
cvta.to.global.u64  %rd4, %rd1;
mul.wide.s32        %rd5, %r1, 8;
add.s64             %rd6, %rd4, %rd5;
ld.global.f64       %fd2, [%rd6];
add.s64             %rd7, %rd3, %rd5;
ld.global.f64       %fd3, [%rd7];
fma.rn.f64          %fd4, %fd2, %fd1, %fd3;
st.global.f64      [%rd7], %fd4;
```

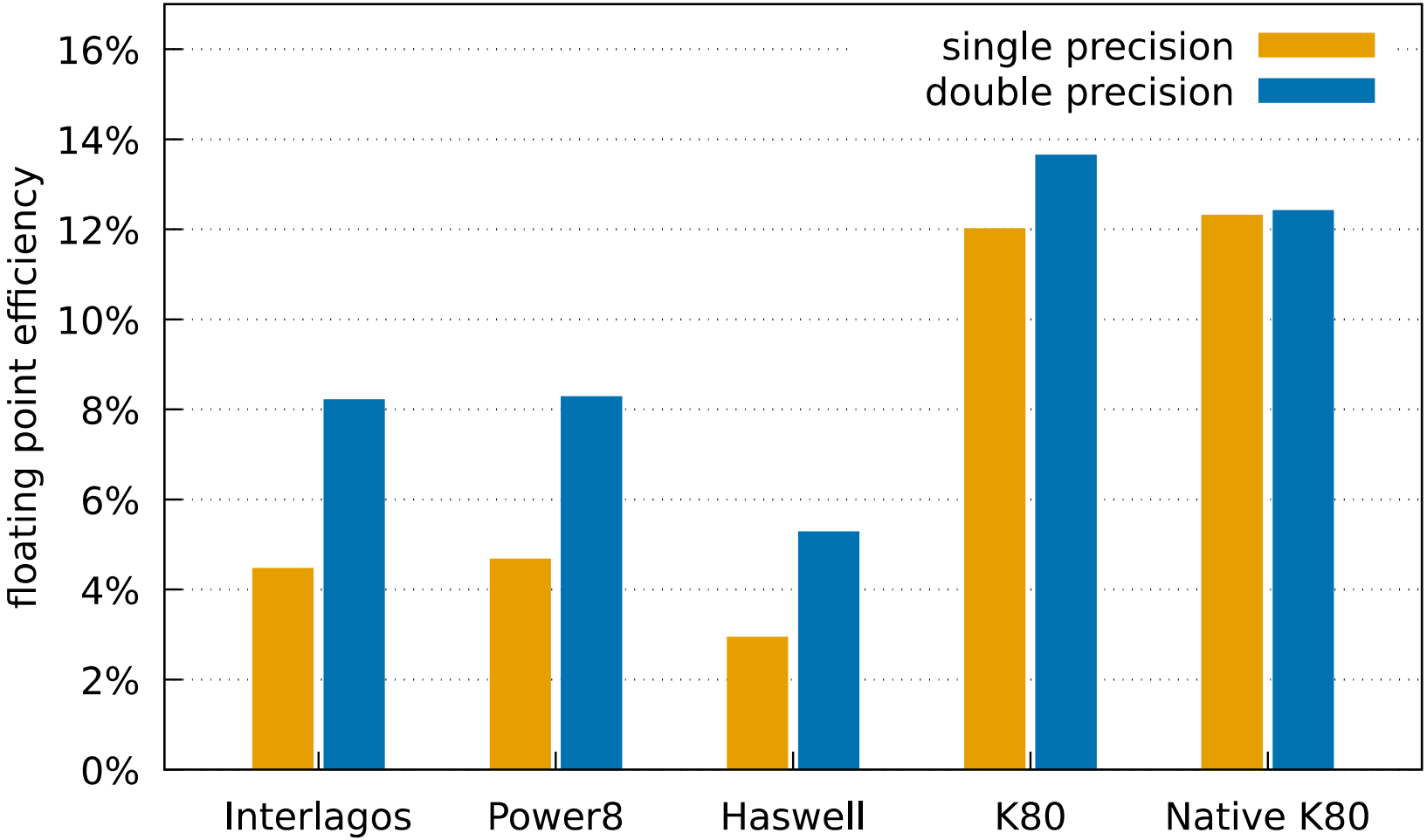
Native CUDA PTX

```
mov.u32      %r3, %ctaid.x;
mov.u32      %r4, %ntid.x;
mov.u32      %r5, %tid.x;
mad.lo.s32   %r1, %r4, %r3, %r5;
setp.ge.s32 %p1, %r1, %r2;
@%p1 bra  BB6_2;

cvta.to.global.u64  %rd3, %rd2;
cvta.to.global.u64  %rd4, %rd1;
mul.wide.s32        %rd5, %r1, 8;
add.s64             %rd6, %rd4, %rd5;
ld.global.nc.f64    %fd2, [%rd6];
add.s64             %rd7, %rd3, %rd5;
ld.global.f64       %fd3, [%rd7];
fma.rn.f64          %fd4, %fd2, %fd1, %fd3;
st.global.f64      [%rd7], %fd4;
```

Single Source Alpaka DGEMM Kernel on Various Architectures



PIConGPU Efficiency on Various Architectures



Clone us from GitHub

https://github.com/ComputationalRadiationPhysics

git clone https://github.com/ComputationalRadiationPhysics/alpaka