Challenges Porting a C++ Template-Metaprogramming Abstraction Layer to Directive-based Offloading

Porting PIConGPU to OpenMP `target` and OpenACC

Jeffrey Kelling[1], Sergei Bastrakov[2], Alexander Debus[2], Thomas Kluge[2], Matt Leinhauser[3,4], Richard Pausch[2], Klaus Steiniger[2], Jan Stephan[4], René Widera[2], Jeff Young[5], Michael Bussmann[4], Sunita Chandrasekaran[3], Guido Juckeland[1]

[1] Department of Information Services and Computing, Helmholtz-Zentrum Dresden-Rossendorf (HZDR)
mailto:j.kelling@hzdr.de, https://www.hzdr.de

[2] Insitute of Radiation Physics, Helmholtz-Zentrum Dresden-Rossendorf (HZDR)
[3] Deptartment of CIS, University of Delaware
[4] Center for Advance Systems Understanding (CASUS)
[5] Georgia Tech, School of Computer Science

October 15, 2021

PIConGPU

github.com/ComputationalRadiationPhysics/picongpu

picongpu.readthedocs.io

- **Open source, fully relativistic, 3D3V, manycore, performance portable PIC code with a single code base for relativistic plasma physics**

- **Implements various numerical schemes, e.g.:**
  - **> *Villasenor-Buneman*, *Esirkepov* and *ZigZag* current deposition**
  - **> *NGP* (0th) to *P4S* (4th) macro particle shape orders**
  - **> *Boris* and *Vay* particle pusher**
  - **> *Yee*, *Lehe and AO-FDTD* field solver**

- **Available self-consistent additions to the PIC cycle, e.g.:**
  - **> QED synchrotron radiation and Bremsstrahlung (photon emission)**
  - **> *Thomas-Fermi* collisional ionization**
  - **> *ADK* and *BSI* field ionization**
  - **> In-situ calculation of coherent and incoherent far field radiation**
  - **> Classical radiation reaction**

- **Tools and diagnostics, e.g.:**
  - **> Extensible selection of plugins for online analysis of particle and field data**
  - **> Scalable I/O for restarts and full output in openPMD using parallel *HDF5* and *ADIOS2***

DRESDEN concept

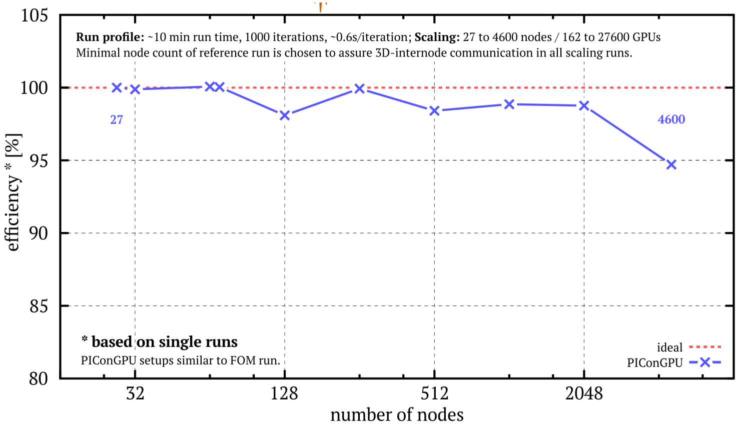
HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

Weak Scaling FOM Case on Summit

PIConGPU is a Frontier CAAR code.

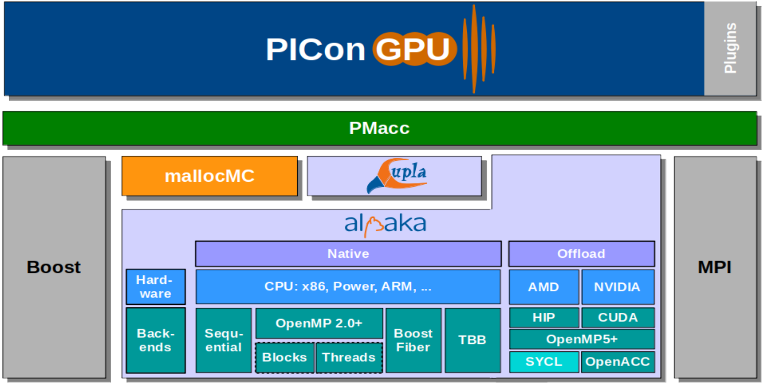
FOM run experimental setup:

- № Iterations: 1000
- Runtime: $\sim$ 10 min
  $\sim$ 0.6 s per iteration
- FOM Science case
- Scaling:
  - 27 $\rightarrow$ 4600 nodes
  - 162 $\rightarrow$ 27 600 GPUs
  - 96–98 % GPU utilization



DRESDEN concept HZDR

PIConGPU Full Software Stack



Huebl, Axel, et al. (2018) Zero Overhead Modern C++ for Mapping to Any Programming Model.
Software Stack updated by René Widera (2020)

Offloading Models

- Vendor Specific, low-level: CUDA, HIP, ...
- Open, low-level: OpenCL, SYCL, ...
- Open, directive-based: OpenMP `target`, OpenACC

Abstraction Layers in C++

- RAJA
- Kokkos
- alpaka

Abstraction Layers in C++

- RAJA
- Kokkos
- alpaka

Why?

Abstraction Layers in C++

- RAJA
- Kokkos
- alpaka

Why?

- Dilemma of choice:
  - Which API to use?
  - Which will be supported throughout the lifetime of the code?
- A future hardware architechture may come with a new programming model...

Abstraction Layers in C++

- RAJA
- Kokkos
- alpaka

Why?

- Dilemma of choice:
  - Which API to use?
  - Which will be supported throughout the lifetime of the code?
- A future hardware architechture may come with a new programming model...

⇒ Important to keep large applications independent of offloading API

- Dependence on abstraction layer less problematic:
  comparatively lightweight, can be maintained by primary application's team

OpenMP `target` and OpenACC

OpenMP `target`

- Extension of OpenMP for accelerator offloading
- Added in verison 4.0
- Aims to provide fine-grained control
- Explicit parallelism

- *#pragma omp target*

OpenMP `target` and OpenACC

OpenMP `target`

- Extension of OpenMP for accelerator offloading
- Added in verison 4.0
- Aims to provide fine-grained control
- Explicit parallelism

- *#pragma omp target*

OpenACC

- Newly developed parallel model specifically for accelerators
- Aims to be descriptive rather than prescriptive
- *Intentionally only pure data parallelism on device*

- *#pragma acc parallel*

Accelerator Execution Hierarchy

CUDA	Alpaka	OpenMP 5.0	OpenACC 3.0	*execution*
grid	grid	(target)	(parallel)	task
block	block	team	gang	undefined
thread	thread	thread	worker	lock-step
—	element	simd	(vector)	vector/seq.

Accelerator Execution Hierarchy

CUDA	Alpaka	OpenMP 5.0	OpenACC 3.0	*execution*
grid	grid	(target)	(parallel)	task
block	block	team	gang	undefined
thread	thread	thread	worker	undefined
—	element	simd	(vector)	vector/seq.

alpaka

- *Header-only C++14 abstraction library for accelerator development*
- Accelerator type passed to device kernels as backend handle

```
template<typename TAcc>
void kernel(const TAcc& acc, ...);
```

⇒ no conditional compilation required for backend selection

- *Header-only C++14 abstraction library for accelerator development*
- Accelerator type passed to device kernels as backend handle

```
template<typename TAcc>
void kernel(const TAcc& acc, ...);
```

⇒ no conditional compilation required for backend selection

- API and feature set modelled after CUDA

host	devices, queues, events, memory management, ...
device	atomics, block-shared memory, block-sync, ...
lib	math, random, ...

- supported backends include:
  sequential, OpenMP, TBB, CUDA, HIP, ...

alpaka: Queues and Tasks

- Compute and memory task objects are placed in queues executing in order

alpaka: Queues and Tasks

- Compute and memory task objects are placed in queues executing in order

```
template<class Functor, class... Args>
struct TaskKernel
{
  TaskKernel(
    WorkDiv workDiv,    // grid size
    Functor functor,    // user functor
    Args ...args );     // user arguments

  void operator() (const DevType& dev) const;

  private:
  WorkDiv m_workDiv;
  Functor m_functor;
  tuple< decay_t<Args...> > m_args;
};
```

alpaka: Kernel

OpenMP target

```
// TaskKernel_Omp5::operator() (...) {
// copy members to local scope, e.g.:
auto args = m_args;
omp_set_num_threads( workdiv.threads );
#pragma omp target
{
# pragma omp teams distribute
  for ( int b = 0; b < workDiv.blocks; ++b )
  {
    // OpenMP backend handle:
    AccOmp5 ctx ( workdiv, b );
#   pragma omp parallel

    {

      apply([&ctx](auto ...args){
          functor ( ctx, args... );
        }, margs);
} } }
```

OpenMP target

```
// TaskKernel_Omp5::operator() (...) {
// copy members to local scope, e.g.:
auto args = m_args;
omp_set_num_threads( workdiv.threads );
#pragma omp target
{
# pragma omp teams distribute
  for ( int b = 0; b < workDiv.blocks; ++b )
  {
    // OpenMP backend handle:
    AccOmp5 ctx ( workdiv, b );
#   pragma omp parallel

    {

      apply([&ctx](auto ...args){
          functor ( ctx, args... );
        }, margs);
} } }
```

OpenACC

```
// TaskKernel_Oacc::operator() (...) {
// copy members to local scope, e.g.:
auto args = m_args;

#pragma acc parallel
{
# pragma acc loop gang
  for ( int b = 0; b < workDiv.blocks; ++b )
  {
    // OpenACC block context:
    CtxBlockOacc ctxBlock (workdiv, b);
#   pragma acc loop worker
    for ( int t = 0; t < workdiv.threads; ++t )
    {
      AccOacc ctx ( ctxBlock, t );
      apply([&ctx](auto ...args){
          functor ( ctx, args... );
        }, margs);
} } }
```

alpaka: Kernel Environment

OpenMP target

- Block-thread index

```cpp
template<>
class GetThreadIdx< AccOmp5 > {
  size_t get ( const AccOmp5& ) {
    return omp_get_thread_num();
  }
};
```

OpenACC

```cpp
template<>
class GetThreadIdx< AccOacc > {
  size_t get ( const AccOacc& ctx ) {
    return ctx.m_threadIdx;
  }
};
```

alpaka: Kernel Environment

OpenMP target

- Block-thread index

```
template<>
class GetThreadIdx< AccOmp5 > {
  size_t get ( const AccOmp5& ) {
    return omp_get_thread_num();
  }
};
```

- Block-level barrier

```
template<>
class SyncBlockThreads< AccOmp5 > {
  void sync ( const AccOmp5& ) {
#   pragma omp barrier
} };
```

OpenACC

```
template<>
class GetThreadIdx< AccOacc > {
  size_t get ( const AccOacc& ctx ) {
    return ctx.m_threadIdx;
  }
};
```

```
template<>
class SyncBlockThreads< AccOacc > {
  void sync ( const AccOacc& acc ) {
  // atomics and spin waits
} };
```

alpaka: Kernel Environment

OpenMP target

- Block-thread index

```
template<>
class GetThreadIdx< AccOmp5 > {
  size_t get ( const AccOmp5& ) {
    return omp_get_thread_num();
  }
};
```

- Block-level barrier

```
template<>
class SyncBlockThreads< AccOmp5 > {
  void sync ( const AccOmp5& ) {
#   pragma omp barrier
} };
```

- Block-shared memory
  - block context contains small-object allocator (∼ 30 kB buffer, configurable)

OpenACC

```
template<>
class GetThreadIdx< AccOacc > {
  size_t get ( const AccOacc& ctx ) {
    return ctx.m_threadIdx;
  }
};
```

```
template<>
class SyncBlockThreads< AccOacc > {
  void sync ( const AccOacc& acc ) {
  // atomics and spin waits
} };
```

alpaka: Memory

- Device (and host) memory are managed via RAII buffer API
- Explicit operations of buffer creation and copy
- No linking between host and device memory ⇒ no use for *data* directives

Alpaka	CUDA	OpenMP 5.0	OpenACC 3.0
`alpaka::allocBuf`	`cudaMalloc`	`omp_target_alloc`	`acc_malloc`

`alpaka::memcpy`	`cudaMemcpy`	`omp_target_memcpy`	`acc_memcpy_to_device`
			`acc_memcpy_from_device`
			`acc_memcpy_device`

~Buf	`cudaFree`	`omp_target_free`	`acc_free`

PIConGPU: Globals and Constants

- Alpaka does not provide and abstraction for global variables.
  - PIConGPU uses one global variable, requiring directives in the code:

```
uint64_t nextId;
#pragma acc declare device_resident(nextId)
#pragma omp declare target(nextId)
```

PIConGPU: Globals and Constants

- Alpaka does not provide and abstraction for global variables.
  - PIConGPU uses one global variable, requiring directives in the code:

```
uint64_t nextId;
#pragma acc declare device_resident(nextId)
#pragma omp declare target(nextId)
```

- PIConGPU's simulation definition is fixed at compile time using `constexpr`.
  - If a constant needs an address at run-time it must be explicitly mapped to the device
  - e.g. for runtime-indexed array, object of which a non-static member function is called in device code

```
constexpr uint64_t constant[] = { 1, 2 }
#pragma acc declare copyin(constant)
#pragma omp declare target(constant)
```

Issues in Standards

- **types containing `static constexpr` data members were not mappable** *(OpenMP `target` (< 5.0))*
  - probably result of a ban on `static` with no regard to `const` in C++

Issues in Standards

- **types containing `static constexpr` data members were not mappable**
  (OpenMP `target` (< 5.0))
  - probably result of a ban on `static` with no regard to `const` in C++
- **mapping of `constexpr` variables with static lifetime (compile-time constants) not implicit**
  (OpenMP `target` / OpenACC)
  - compiler knows which constants are used and there is no abiguity about sequence of copy $\Rightarrow$ should be implicit

Issues in Standards

- **types containing `static constexpr` data members were not mappable**
  (OpenMP `target` (< 5.0))
  - probably result of a ban on `static` with no regard to `const` in C++
- **mapping of `constexpr` variables with static lifetime (compile-time constants) not implicit**
  (OpenMP `target` / OpenACC)
  - compiler knows which constants are used and there is no abiguity about sequence of copy ⇒ should be implicit
- **missing gang-level barrier**
  (OpenACC)
  - a barrier would not agree with pure data-parallel philosophy
  - no explicit control over number of workers ⇒ makeshift barrier can dead-lock or not work

Issues in Standards

- **types containing `static constexpr` data members were not mappable**
  (OpenMP `target` (< 5.0))
  - probably result of a ban on `static` with no regard to `const` in C++
- **mapping of `constexpr` variables with static lifetime (compile-time constants) not implicit**
  (OpenMP `target` / OpenACC)
  - compiler knows which constants are used and there is no abiguity about sequence of copy ⇒ should be implicit
- **missing gang-level barrier**
  (OpenACC)
  - a barrier would not agree with pure data-parallel philosophy
  - no explicit control over number of workers ⇒ makeshift barrier can dead-lock or not work
- **`std::tuple` implementations are not required to be trivially copyable if all component types are**
  (C++)

  ⇒ no `std::tuple` is formally mappable

Tested Compilers

target:	OpenMP `target` x86	OpenMP `target` hsa	OpenMP `target` nvptx	OpenACC x86	OpenACC nvptx
GCC $\geq$ 9	☐			☐	
Clang $\geq$ 10	☐	☐	☐		
AOMP $\approx$ 0.7	☐	☐			
ROC Clang $=$ 4.3.0		☐			
IBM XL $=$ 16.1.1-5			☐		
NVHPC $\geq$ 19.3				☐	☐

- All listed compilers showed major roadblocks in initial tests.
- Followed only updates of two compilers with fastest development speed:
  - Clang (git main) for OpenMP `target`
  - NVHPC (releases) for OpenACC

Compiler Issues I

Main complication turned out to be a lack of tested compiler support:

- OpenMP 5.0 / OpenACC 3.0 not fully supported anywhere. E.g:

  GCC types with `static constexpr` not mappable (very strict interpretation of OpenMP 4.5)
  $\Rightarrow$ porting PIConGPU impossible

Compiler Issues I

Main complication turned out to be a lack of tested compiler support:

- OpenMP 5.0 / OpenACC 3.0 not fully supported anywhere. E.g:

  GCC types with `static constexpr` not mappable (very strict interpretation of OpenMP 4.5)
  $\Rightarrow$ porting PIConGPU impossible

- Internal Compiler Errors (ICE) happen when directives meet C++

Compiler Issues I

Main complication turned out to be a lack of tested compiler support:

- OpenMP 5.0 / OpenACC 3.0 not fully supported anywhere. E.g:

  GCC types with `static constexpr` not mappable (very strict interpretation of OpenMP 4.5)
  $\Rightarrow$ porting PIConGPU impossible

- Internal Compiler Errors (ICE) happen when directives meet C++
- Invalid use or not-implemented features can trigger ICE instead of compiler error

Compiler Issues I

Main complication turned out to be a lack of tested compiler support:

- OpenMP 5.0 / OpenACC 3.0 not fully supported anywhere. E.g:

  GCC types with `static constexpr` not mappable (very strict interpretation of OpenMP 4.5)
  $\Rightarrow$ porting PIConGPU impossible

- Internal Compiler Errors (ICE) happen when directives meet C++
- Invalid use or not-implemented features can trigger ICE instead of compiler error
- Runtime errors, like incorrect data sharing, atomics not doing what they should

Compiler Issues II

- Focussed main development and testing on Alpaka's test suite and examples, rather than PIConGPU

⇐ Smaller applications with limited scope may not get stuck on the same bugs

Compiler Issues II

- Focussed main development and testing on Alpaka's test suite and examples, rather than PIConGPU

⇐ Smaller applications with limited scope may not get stuck on the same bugs

- When code compiles but does not work due to compiler bug correctness of code must be shown—hard when no second compiler compiles the code

Compiler Issues II

- Focussed main development and testing on Alpaka's test suite and examples, rather than PIConGPU

⇐ Smaller applications with limited scope may not get stuck on the same bugs

- When code compiles but does not work due to compiler bug correctness of code must be shown— hard when no second compiler compiles the code

⇒ Sometimes needed compiler developers to run our complete code through their compiler to debug issues without small reproducer

Results: alpaka VectorAdd

```
auto bufHostA(alpaka::allocBuf<uint32_t, Idx>(devHost, extent)); //... bufHostB(...), bufHostC(...);
// init bufHost* ...
auto bufAccA(alpaka::allocBuf<uint32_t, Idx>(devAcc, extent)); //... bufAccB(...), bufAccC(...);
alpaka::memcpy(queue, bufAccA, bufHostA, extent); // ...

auto const taskKernel = alpaka::createTaskKernel<Acc>(workDiv,
  [](const auto& acc, const uint32_t* A, const uint32_t* B, uint32_t* C, size_t N){
      //...
      for(TIdx i(threadFirstElemIdx); i < threadLastElemIdxClipped; ++i)
          C[i] = A[i] + B[i];
  }, alpaka::getPtrNative(bufAccA), alpaka::getPtrNative(bufAccB), alpaka::getPtrNative(bufAccC), N);

alpaka::enqueue(queue, taskKernel);
alpaka::memcpy(queue, bufHostC, bufAccC, extent);
alpaka::wait(queue); // check result against host computation
```

Results: alpaka VectorAdd

```
auto bufHostA(alpaka::allocBuf<uint32_t, Idx>(devHost, extent)); //... bufHostB(...), bufHostC(...);
// init bufHost* ...
auto bufAccA(alpaka::allocBuf<uint32_t, Idx>(devAcc, extent)); //... bufAccB(...), bufAccC(...);
alpaka::memcpy(queue, bufAccA, bufHostA, extent); // ...

auto const taskKernel = alpaka::createTaskKernel<Acc>(workDiv,
  [](const auto& acc, const uint32_t* A, const uint32_t* B, uint32_t* C, size_t N){
        //...
        for(TIdx i(threadFirstElemIdx); i < threadLastElemIdxClipped; ++i)
            C[i] = A[i] + B[i];
  }, alpaka::getPtrNative(bufAccA), alpaka::getPtrNative(bufAccB), alpaka::getPtrNative(bufAccC), N);

alpaka::enqueue(queue, taskKernel);
alpaka::memcpy(queue, bufHostC, bufAccC, extent);
alpaka::wait(queue); // check result against host computation
```

	Clang Main		ROC Clang	NVHPC 21.7	
	x86	hsa	hsa	x86	nvptx
compile	✓	✓	✓	✓	✓
run	✓	memory error	✓	✓	✓

DRESDEN concept HZDR

Results: alpaka Test Suite

- Suite of tests also used in alpaka's CI
- Battery of test cases for each aspect of a backend: kernels, memory, atomics, ...
- Using Catch2 ⇒ more TMP, harder for compilers to succeed.

[1] only local installation, `nvlink error : Duplicate weak parameter bank for ...` when using NVIDIA docker image in CI

Results: alpaka Test Suite

- Suite of tests also used in alpaka's CI
- Battery of test cases for each aspect of a backend: kernels, memory, atomics, ...
- Using Catch2 ⇒ more TMP, harder for compilers to succeed.

	Clang Main		ROC Clang	NVHPC 21.7		GCC 11
	x86	hsa	hsa	x86	nvptx	x86
compile	✓	most	slow, linker hangs	✓	✓[1]	most
run	✓	memory error		✓	✓	✗

[1] only local installation, `nvlink error : Duplicate weak parameter bank for ...` when using NVIDIA docker image in CI

Results: PIConGPU

	Clang Main	NVHPC 21.7	
	x86	x86	nvptx
compile	✓	✓	✓
run	✓	✓	✗

Outlook

- OpenMP `target` and OpenACC compiler ecosystems still rather unstable when it comes to C++
- OpenACC is too strict about data parallelism to port existing codes which do not adhere to this pattern

Outlook

- OpenMP `target` and OpenACC compiler ecosystems still rather unstable when it comes to C++
- OpenACC is too strict about data parallelism to port existing codes which do not adhere to this pattern

- Our OpenMP `target` and OpenACC backends are, to our knowledge, complete, though we cannot actually test and debug them completely
- Will follow and try to push future compiler development

Acknowledgments

- Mathew Colgrove (NVIDIA) and NVHPC for helping to debug compiler and code issues
- Ron Liberman (AMD) and SPEC High Performance group for advice and testing PIConGPU

Thank You.

OpenMP `target` and OpenACC: Directives

	OpenMP `target`	OpenACC
execution	*omp target*	*acc parallel*
	omp teams distribute	*acc loop gang*
	omp parallel for	*acc loop worker*

OpenMP `target` and OpenACC: Directives

	OpenMP `target`	OpenACC
execution	*omp target*	*acc parallel*
	omp teams distribute	*acc loop gang*
	omp parallel for	*acc loop worker*
memory	*omp target data map (...)*	*acc data copy...*
	omp declare target (...)	*acc declare (...)*

OpenMP `target` and OpenACC: Directives

	OpenMP `target`	OpenACC
execution	*omp target*	*acc parallel*
	omp teams distribute	*acc loop gang*
	omp parallel for	*acc loop worker*
memory	*omp target data map (...)*	*acc data copy...*
	omp declare target (...)	*acc declare (...)*
atomics	*omp atomic*	*acc atomic*
lock	*omp critical*	—
sync threads	*omp barrier*	—

Results: alpaka HelloWorld

```
alpaka::exec<Acc>( queue, workDiv,
   [] (const auto& acc) {
      const auto gidx = alpaka::getIdx<alpaka::Grid, alpaka::Threads>(acc);
      const auto gext = alpaka::getWorkDiv<alpaka::Grid, alpaka::Threads>(acc);

      const auto lgidx = alpaka::mapIdx<1u>( gidx, gext );

      printf("[z:%u, y:%u, x:%u][linear:%u] Hello World\n", gidx[0u], gidx[1u], gidx[2u], lgidx[0u] );
   } );
alpaka::wait(queue);
```

Results: alpaka HelloWorld

```cpp
alpaka::exec<Acc>( queue, workDiv,
  [] (const auto& acc) {
      const auto gidx = alpaka::getIdx<alpaka::Grid, alpaka::Threads>(acc);
      const auto gext = alpaka::getWorkDiv<alpaka::Grid, alpaka::Threads>(acc);

      const auto lgidx = alpaka::mapIdx<1u>( gidx, gext );

      printf("[z:%u, y:%u, x:%u][linear:%u] Hello World\n", gidx[0u], gidx[1u], gidx[2u], lgidx[0u] );
  } );
alpaka::wait(queue);
```

	Clang Main		ROC Clang	NVHPC 21.7	
	x86	hsa	hsa	x86	nvptx
compile	✓	no c-lib	✓	✓	✓
run	✓		✓	✓	✓