Towards Exascale with IPPL Version 2.0

OPAL developers meeting

Matthias Frey[1] Sriramkrishnan Muralikrishnan[2] Andreas Adelmann[2]

[1]Mathematical Institute, University of St Andrews, UK

[2]Paul Scherrer Institute, PSI Villigen, CH

19 Jan 2022

IPPL 2.0 developers

Goals of IPPL

- **Portability** across serial, distributed, and parallel architectures
- Development of reusable, cross-problem-domain components to **enable rapid application development**
- **High efficiency** for kernels and components relevant to scientific simulation
- Framework **design and development driven by applications** from a diverse set of scientific problem domains
- **Shorter time** from problem inception **to working parallel simulations**

Performance Portable Abstraction Layers

library	**lang**	**no. contr.**[3]	**license**	**repository**
Kokkos	`C++`	99	BSD 3-Clause	`https://github.com/kokkos`
alpaka	`C++`	28	MPL-2.0	`https://github.com/alpaka-group/alpaka`
Thrust	`C++`	65	Apache License 2.0	`https://github.com/thrust/thrust`
RAJA	`C++`	42	BSD 3-Clause	`https://github.com/LLNL/RAJA`

backend	*Kokkos*	*alpaka*	*Thrust*	*RAJA*
OpenMP	✓	✓	✓	✓
Pthreads	✓	✗	✗	✗
`C++11` threads	✗	✓	✗	✗
CUDA	✓	✓	✓	✓
TBB	✗	✓	✓	⚠
HPX	✓	✗	✗	✗
HIP[4]	✓	✓	✗	✓
SYCL[5]	✓	✗	✗	⚠

programming	*Kokkos*	*alpaka*	*Thrust*	*RAJA*
memory management	✓	✓	✓	✗
complexity	medium/high	high	low/medium	medium/high

[3]Number of contributors on 18 Jan 2022

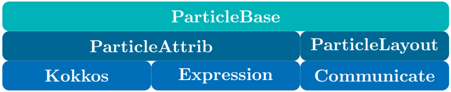
[4]Interface for NVIDIA and AMD GPUs

[5]Interface for NVIDIA, AMD and Intel CPUs and GPUs

Goals in the development of IPPL 2.0

- Enable **performance portability with Kokkos**
  (i.e. replace field and particle containers with Kokkos data structures)
- Upgrade to `C++17` **standard**
- Keep **expression templates** for particles and fields
- **Keep** changes to the **user interface** to a minimum
- **Simplify code**

Particles in IPPL 2.0



- **Attributes:**
  - Struct of `Kokkos::Views`
  - Expression templates
  - Easily added to application
- **Communication:**
  - Particle layout classes
  - De-/serialize `Kokkos::View<char*>`
  - Pre-allocated buffers

```cpp
using namespace ippl;

template<class PLayout>
struct Bunch
: public ParticleBase<PLayout> {
    Bunch(PLayout& playout)
    : ParticleBase<PLayout>(playout)
    {
        // add application attributes
        this->addAttribute(R);
        this->addAttribute(V);
        this->addAttribute(mass);
        this->addAttribute(charge);
    }

    ~Bunch() { }

    ParticleAttrib<double> mass, charge;

    ParticleAttrib<Vector<double>> R, V;
};

// compiles to single Kokkos kernel
bunch->R = bunch->R + dt * bunch->V;
```

Fields in IPPL 2.0

- **Scalar/Vector fields:**
  - `Kokkos::Views`
  - Expression templates
- **Interface to heFFTe**[6]
- **Communication:**
  - Field layout
  - Distribution of local domains globally known
  - De-/serialize `Kokkos::View<char*>`
  - Pre-allocated buffers

```
using namespace ippl;

constexpr unsigned int dim = 3;

int pt = 256;
Index I(pt);
NDIndex<dim> owned(I, I, I);

// specifies SERIAL, PARALLEL dimensions
e_dim_tag decomp[dim] = {PARALLEL,
                         PARALLEL,
                         PARALLEL};

FieldLayout<dim> layout(owned, decomp);

double dx = 1.0 / double(pt);
Vector<double, dim> hx = {dx, dx, dx};
Vector<double, dim> origin = {0, 0, 0};

UniformCartesian<double, dim>
    mesh(owned, hx, origin);

Field<double, dim> field(mesh, layout);
```

[6] Ayala A., Tomov S., Haidar A., Dongarra J. (2020) heFFTe: Highly Efficient FFT for Exascale. ICCS 2020. https://doi.org/10.1007/978-3-030-50371-0_19

Expression templates

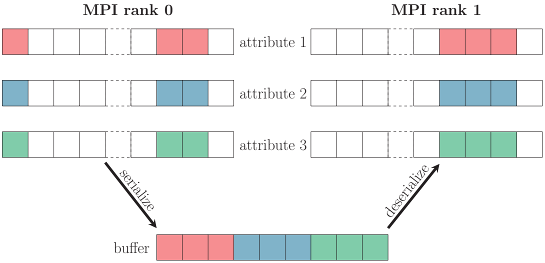
- Avoids temporary objects in mathematical expressions
- Reduces number of Kokkos kernels
- Assignment operator evaluates expression
- Available for particles and fields
- Supported:
  - Binary operations: $+, -, *, /$
  - Comparison operations: $<, \leq, >, \geq, ==, !=$
  - Bitwise operations
  - Many operations of `cmath` header

```
template<typename T, class... Properties>
template <typename E, size_t N>
ParticleAttrib<T, Properties...>&
ParticleAttrib<T, Properties...>::operator=(
    detail::Expression<E, N> const& expr)
{
    using capture_type =
        detail::CapturedExpression<E, N>;

    capture_type expr_ =
        reinterpret_cast<
            const capture_type&>(expr);

    Kokkos::parallel_for(
        "ParticleAttrib::operator=()",
        dview_m.extent(0),
        KOKKOS_CLASS_LAMBDA(const size_t i) {
            dview_m(i) = expr_(i);
    });

    return *this;
}
```

Communication

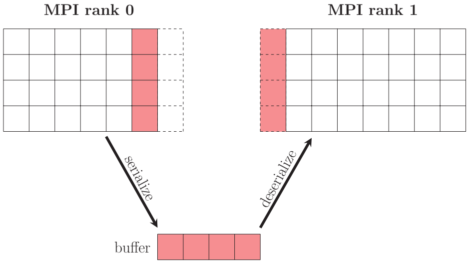
Particles:

- Locate particles to send
- Pack attributes into buffer
- Send buffer to receiver
- Unpack attributes

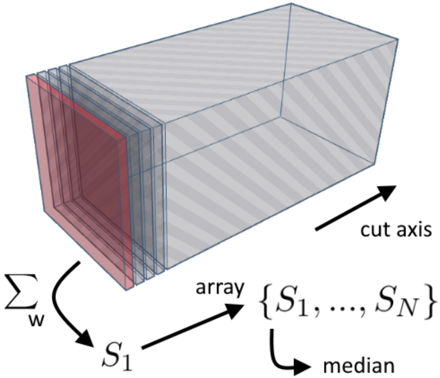


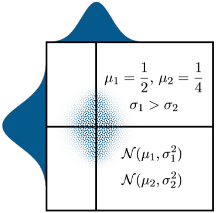
Fields:

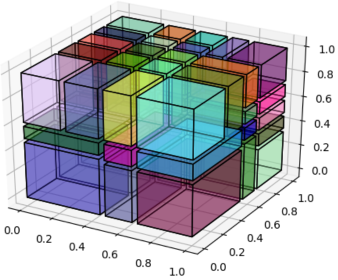
- Get grid intersection
- Pack intersection into buffer
- Send buffer to receiver
- Unpack intersection



1. Interpolate particles to grid (weights only)
2. Slice cut axis and evaluate sum of weights S_i
3. Evaluate median over all S_i
4. Cut (sub-)domain at median



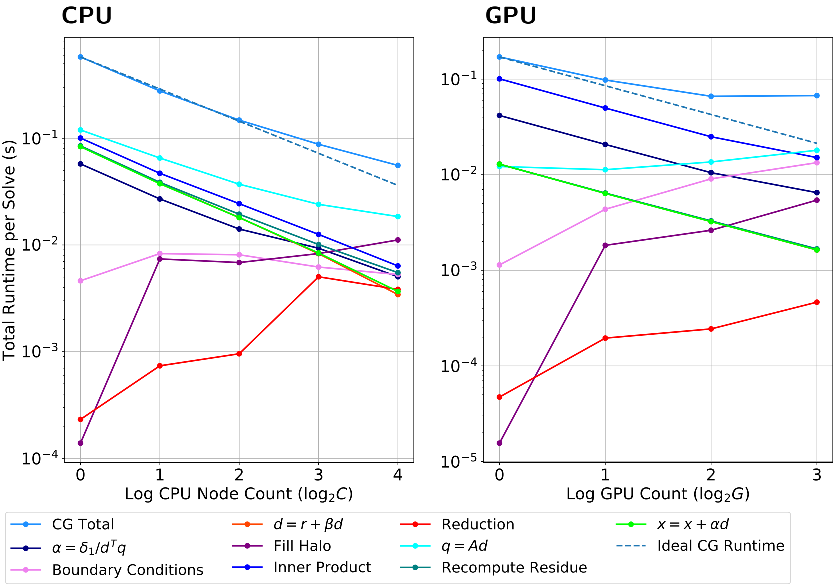




Example with 64 MPI ranks

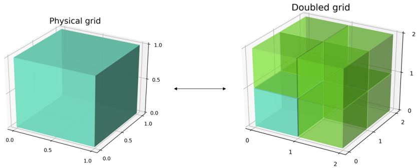
Conjugate Gradient Solver – Timings for 512^3 grid

ETH Bachelor thesis, Alessandro Vinciguerra

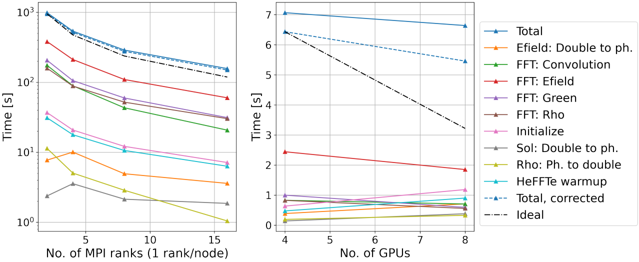


Algorithms:

- Hockney-Eastwood[7]
- Vico-Greengard[8]



Hockney-Eastwood, 256^3 grid



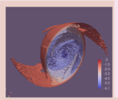
[7] Hockney, R. and Eastwood, J. (1988). Computer Simulation Using Particles. CRC Press.

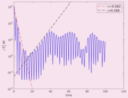
[8] Vico, F., Greengard, L., and Ferrando, M. (2016). Fast convolution with free-space green's functions. Journal of Computational Physics, 323:191–203.

ALPINE: A set of portable pLasma physics Particle-in-cell mINi-apps for Exascale

Two-stream Instability | Penning trap | Landau damping

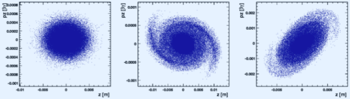
- Light weight codes
- Proxy for real applications
- For implementing new algorithms
- Testing new implementations
- Reference results ensure correctness





OPAL: Object oriented Parallel Accelerator Library

Open source C++ Library for particle accelerators being developed by twelve core developers across seven institutes

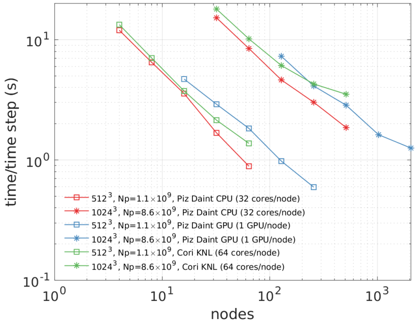


Source: OPAL web page

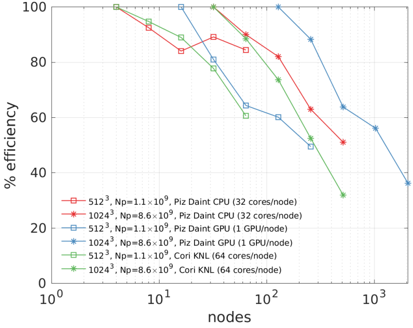
Independent Parallel Particle Layer (IPPL) v 2.0

- FFT based Poisson Solver
- CG
- heFFTe
- D-Operators
- Interpolation
- Fields
- Mesh
- Particles
- Load Balancing
- Domain Decomp.
- Communication
- Buffer factory
- MPI
- Kokkos

Linear Landau damping: Strong scaling

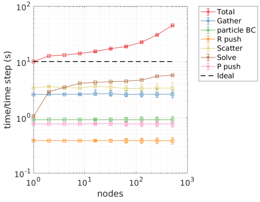


Total time

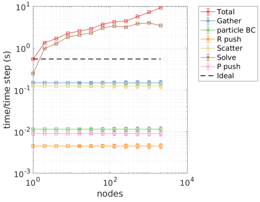


Efficiency

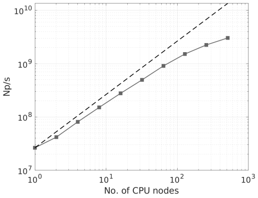
Linear Landau damping: Weak scaling

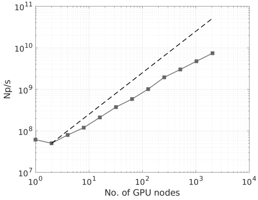


Piz Daint CPUs



Piz Daint GPUs





- For GPUs: 256×128^2 grid and 8 particles per cell is the base case for 1 node/GPU. The max. grid size and particles are 2048^3 and $Np \approx 69$ billion at 2048 GPUs
- For CPUs: 512×256^2 grid and 8 particles per cell is the base case for 1 node. The max. grid size and particles are 4096×2048^2 and $Np \approx 138$ billion at 512 nodes = 16,384 cores

Ongoing and Future work

- Improve load balancing and memory usage
- Use IPPL 2.0 in OPAL ("partial rewrite OPAL required")

Thank you for your attention!

Thanks to

- Alessandro Vinciguerra
- Sonali Mayani
- Michael Ligotino