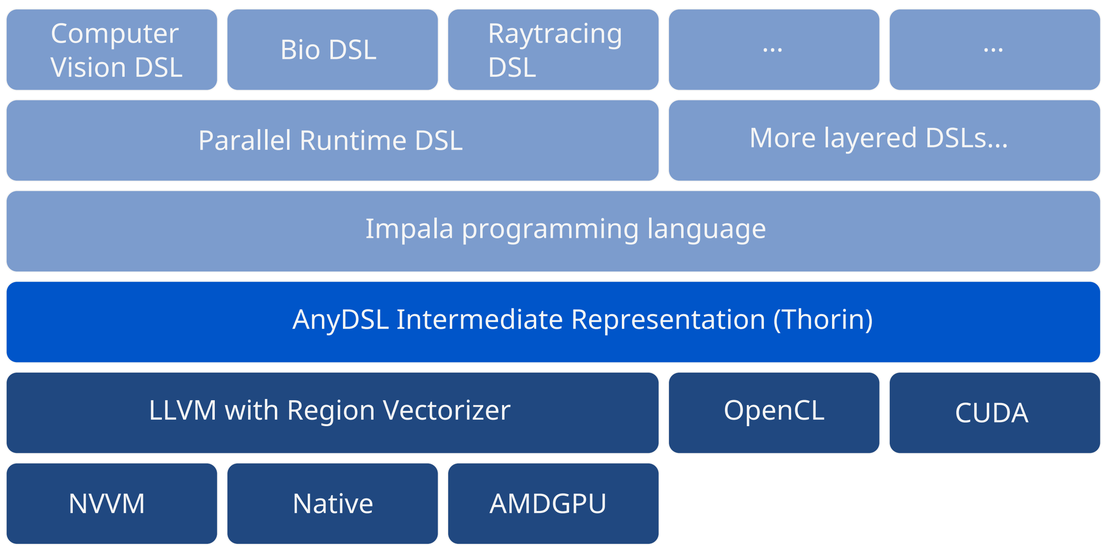
Jonas Schmitt[1] **Harald Köstler**[1] **Jan Eitzinger**[2] **Richard Membarth**[3] **Arsène Pérard-Gayot**[3]

[1]Chair for System Simulation (LSS) and [2]Regional Computing Center Erlangen (RRZE), Universität Erlangen-Nürnberg

[3]German Research Center for Artificial Intelligence (DFKI), Universität des Saarlandes

Unified Code Generation for the Parallel Computation of Pairwise Interactions using Partial Evaluation

AnyDSL - A Framework for Rapid Development of Domain-Specific Libraries



The AnyDSL Approach to Code Generation

- Uniform syntax for the static and dynamic parts of a program
- Code generation is triggered through **Partial Evaluation**
- Typesafe template metaprogramming without additional syntax

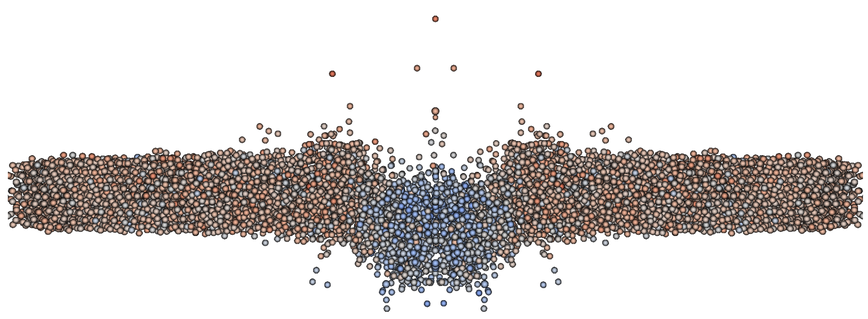
Template Metaprogramming in C++

```
template <int N> struct Factorial {
  enum{
    value = N*Factorial<N-1>::value
  };
};
template <> struct Factorial<0> {
  enum {value = 1};
};
```

Partial Evaluation in Impala

```
// by supplying the @-annotation
// the compiler will evaluate this function
// at every call-side where n is available
fn factorial(@n: i32) -> i32 {
  if n == 0 {1}
  else {n * factorial(n - 1)}
}
```

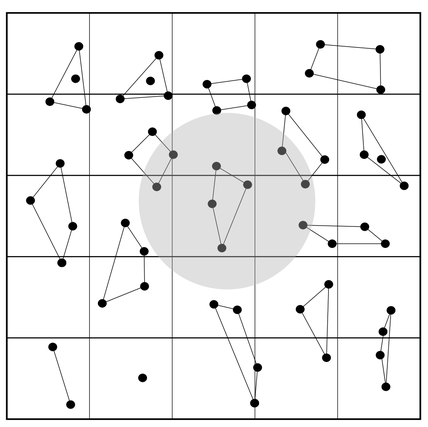
Molecular Dynamics



- Simulation of the trajectories of a large number of particles based on their interactions
- The computation of short-range interactions is an important use case in many simulations
- Most implementations employ a combination of **cell decomposition** and **neighbor lists**

Computing Pairwise Interactions Efficiently on Modern Architectures

- In 2013 Páll and Hess presented an adaption of the neighbor list scheme to modern SIMD and GPU architectures
- Particles are not treated individually but as a cluster of *N* particles
- Interactions are computed between clusters
- Choosing *N* according to the SIMD/SIMT width enables data parallel computation



Kernel Generation through Partial Evaluation

- The execution of a certain computation on a system of particles can be expressed as the following higher-order function:

```
fn execute(particles: Particles, kernel: fn(i32, i32, i32) -> ()) -> ();
```

- By partially evaluating **execute** with respect to its second argument, code generation is triggered
- All details about the target platform are hidden within its implementation
- To generate code for different platforms, different implementations must be provided

Kernel Generation on the CPU

- The AnyDSL runtime library provides functionality for automatic **parallelization** and **vectorization** on the CPU

```
fn execute(particles: Particles, @kernel: fn(i32, i32, i32) -> ()) -> () {
  // Thread-parallel execution
  parallel(get_number_of_threads(), i, 0, particles.number_of_clusters, |ci| {
    let cluster_size = get_cluster_size();
    let begin = ci * cluster_size;
    // Vectorization using RV
    vectorize(cluster_size, get_alignment(), 0, cluster_size, |i| {
      let pi = begin + i;
      kernel(pi, ci, cluster_size);
    });
  });
}
```

Kernel Generation on the GPU

- For execution on GPU hardware, the **accelerator struct** can be employed, which supports CUDA, NVVM and OpenCL as backend

```
fn execute(particles: Particles, @kernel: fn(i32, i32, i32) -> ()) -> () {
  let acc = get_accelerator(device_id);
  let grid = (particles.number_of_clusters * get_cluster_size(), 1, 1);
  let block = (get_cluster_size(), 1, 1);
  acc.exec(grid, block, |bid, bdim, gid| {
    let (gidx, _, _) = gid;
    let (bidx, _, _) = bid;
    let (bdimx, _, _) = bdim;
    kernel(gidx(), bidx(), bdimx());
  });
  acc.sync();
}
```

Single-Core Performance in FLOPS/cycle

- AnyDSL: LLVM version 5.0.1 with RV for vectorization, -O3, -march=native
- MiniMD: Intel C compiler version 18 with -O3, -xHost, -qopt-zmm-usage=high (SKL)

Processor	AnyDSL	MiniMD
Skylake	5.816 (AVX512)	3.618 (AVX512)
Broadwell	2.928 (AVX2)	1.695 (AVX2)
Ivy Bridge	2.103 (AVX)	1.034 (AVX)

Acceleration on the GPU

- CPU test platform: Intel Xeon E3-1275 v5 with four cores
- GPU test platform: NVIDIA GTX 1080, AnyDSL Backend: NVVM, Cluster size: 32
- Double-precision floating-point computations
- The generated GPU code runs around 5 times faster

Particles	AnyDSL (AVX2)	AnyDSL (GPU)
100 000	1044.39 ms	194.101 ms
500 000	4300.2 ms	826.627 ms
1 000 000	7652.97 ms	1684.18 ms
2 000 000	15014.7 ms	3294.85 ms

https://www10.cs.fau.de **jonas.schmitt@fau.de**