



University of
Strathclyde
Science

oxDNA-Kokkos in LAMMPS

Delivering Performance-Portable, Multi-GPU
Coarse-Grained DNA Modelling

HPC-AI Advisory Council
8th Annual UK Conference

PGR/PhD | Lewis Russell
(Presenting)

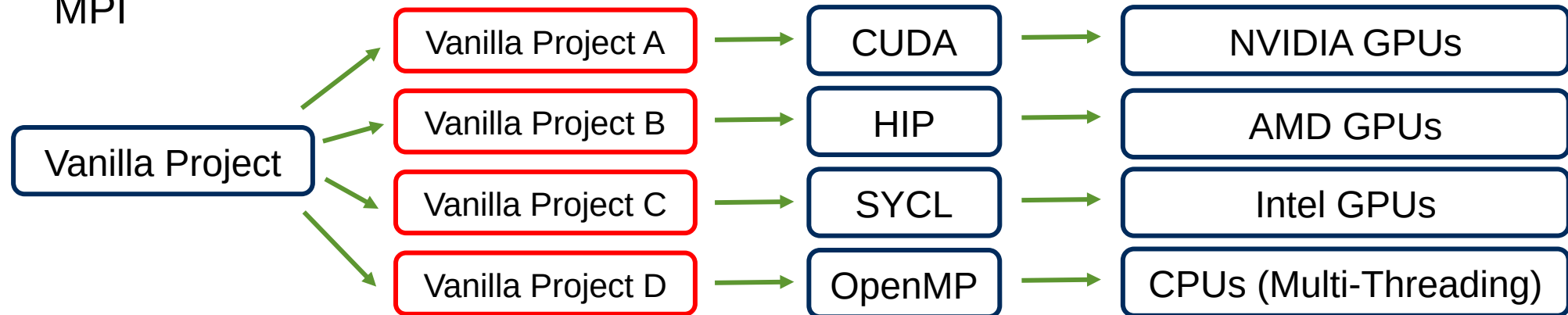
Supervisor | Dr. Oliver Henrich

Motivation

- 8 out of top 10 supercomputers use GPUs | (top500.org/lists/top500/2026/06/)
- At UoStrathclyde, on ARCHIE-WeSt | (www.archie-west.ac.uk/information/archie-specification-new/)
 - Eight NVIDIA A100 40GB GPUs (PCIe)
 - Four 4-way NVIDIA A100SXM 80GB GPUs
- **Special Code Required** for GPUs and many-core CPUs, beyond regular C++ and MPI

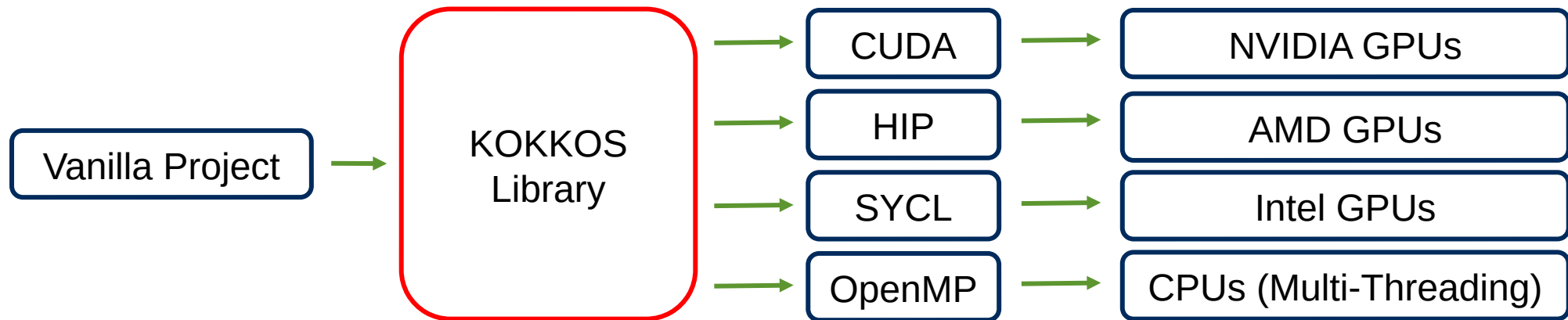
Motivation

- 8 out of top 10 supercomputers use GPUs | (top500.org/lists/top500/2026/06/)
- At UoStrathclyde, on ARCHIE-WeSt | (www.archie-west.ac.uk/information/archie-specification-new/)
 - Eight NVIDIA A100 40GB GPUs (PCIe)
 - Four 4-way NVIDIA A100SXM 80GB GPUs
- **Special Code Required** for GPUs and many-core CPUs, beyond regular C++ and MPI





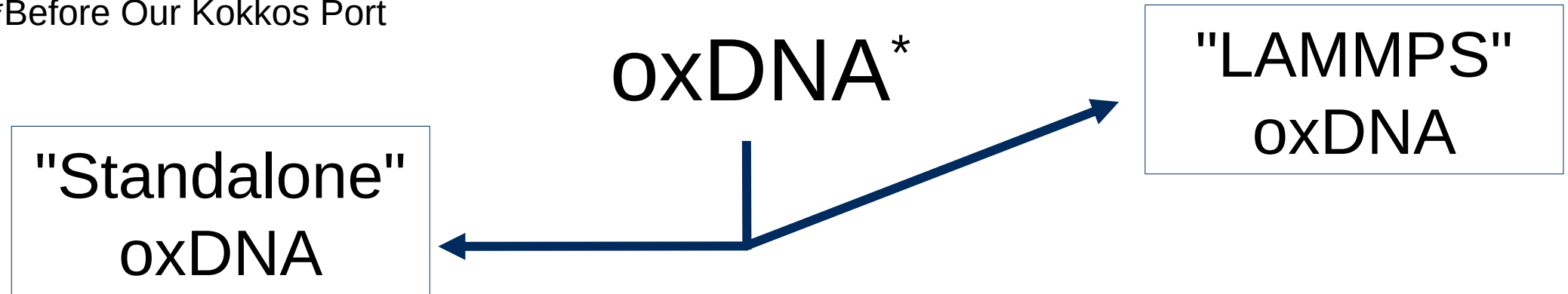
Developed by SNL, ORNL, ANL, LANL, CSCS (Switzerland), CEA (France), ULiège (Belgium)
Common C++ interface over various parallel programming models: CUDA, POSIX Threads, ...
Kokkos Ecosystem | <https://kokkos.org>



Similar Frameworks:

- RAJA (LLNL) | C++, more micro-optimising options, gradual porting
- SYCL | language-based, more complex (explicit) task queues, compilers can be complex

*Before Our Kokkos Port



*Before Our Kokkos Port

"Standalone"
oxDNA

1 CPU Core,
1 Nvidia
GPU

oxDNA*

"LAMMPS"
oxDNA



*Before Our Kokkos Port

"Standalone"
oxDNA

1 CPU Core,
1 Nvidia
GPU

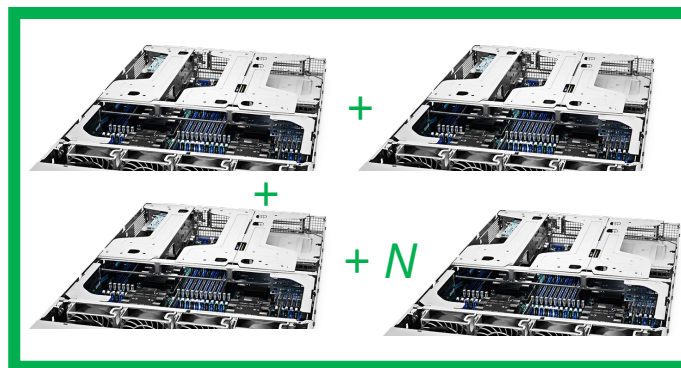


oxDNA*

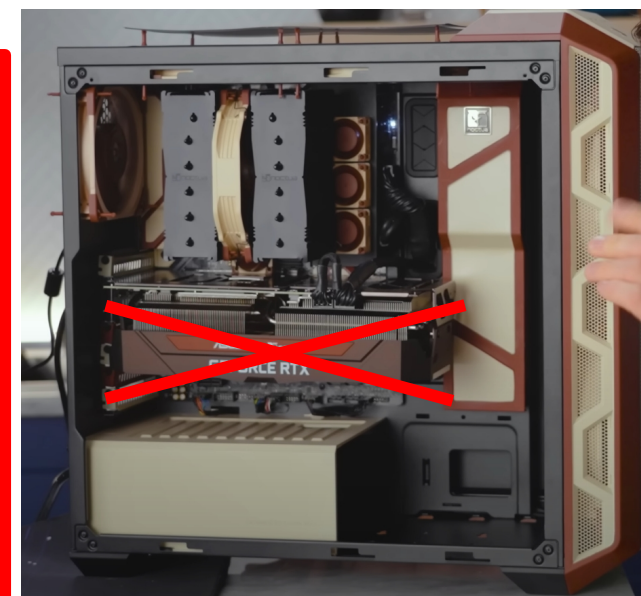


"LAMMPS"
oxDNA

Many
CPUs



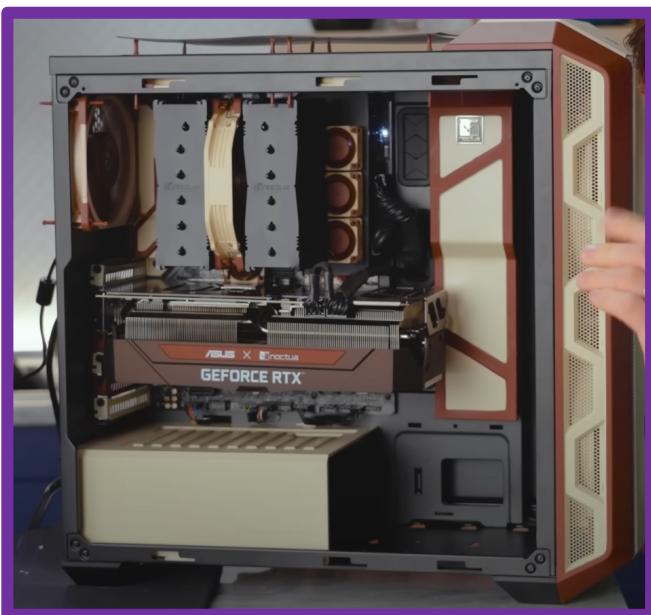
No GPUs



*Before Our Kokkos Port

"Standalone"
oxDNA

1 CPU Core,
1 Nvidia
GPU



oxDNA*

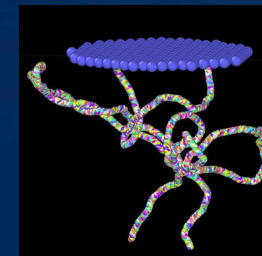
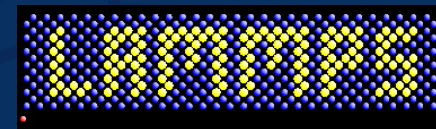
"LAMMPS"
oxDNA

Many
GPUs

LAMMPS -> 'nice' features

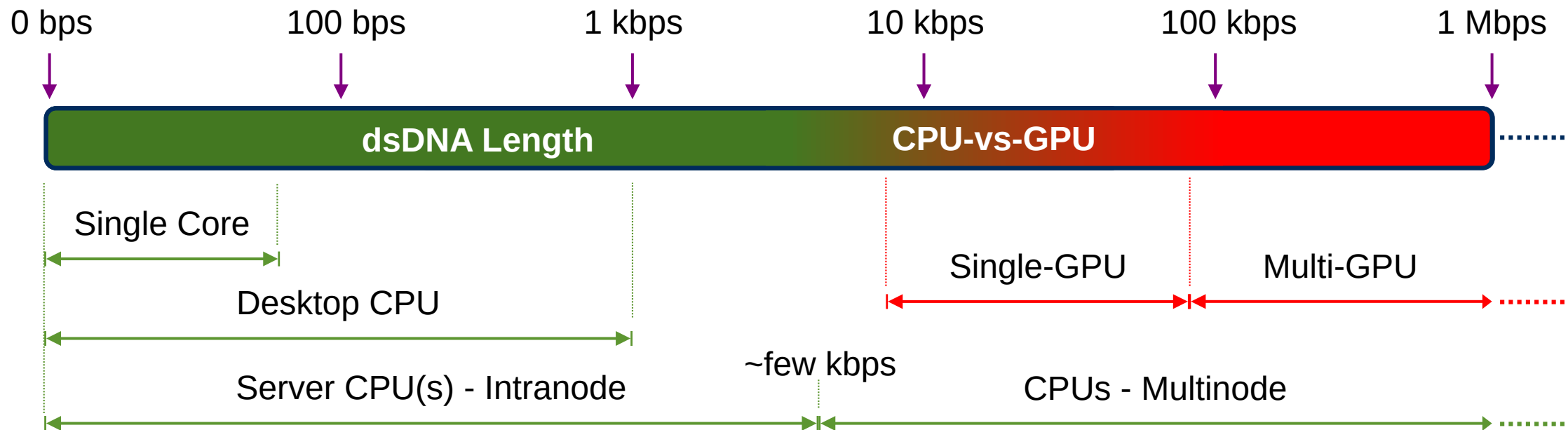
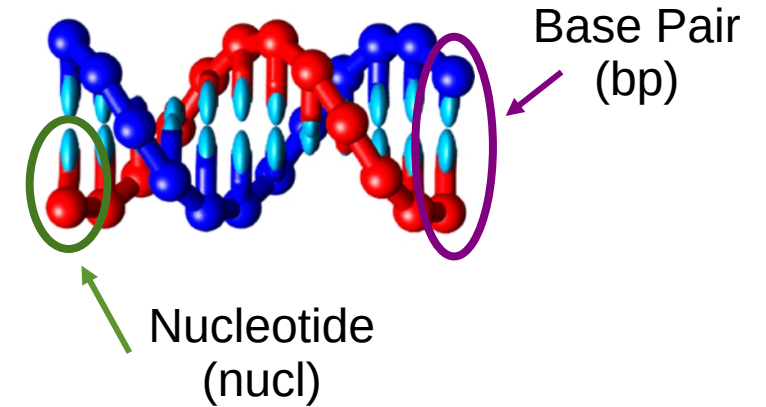
- ~100 General cmds
- ~100s 'Fix' cmds (timestepping edits)
- ~100s 'Compute' cmds
- 1000s of pair/bond/etc interactions
- IWYU Principles – 100 Packages to choose from, only need to build what you need.

<https://docs.lammps.org/Commands.html>



oxDNA | What Hardware When

- CPU for smaller systems, **~50bps per core**
- GPU for bigger systems, **~10kbps+ per GPU**
- **Big Win** for GPU >~ **50kbps** | easily >**50x Perf/W**



How to Port to KOKKOS

Very Simple Answer |

- If it's within a parallel region, convert the C++ data type to KOKKOS data type (scalars fine)
- Lots of *KOKKOS Views*, essentially exotic and advance polymorphic versions of C++ pointers
- Determine the parallel dispatch flavour:
 - `parallel_for` | run a loop in parallel
 - `parallel_reduce` | run a loop in parallel, and compute one or more global reductions
 - `parallel_scan` | prefix reduction across all threads
- Encapsulate the upper most iteration of the loop body, kernalising out in one of **three** ways:
 - C++ LAMBDA | Inlined and anonymous. Can be useful for short kernels.
 - Functor | Most general but take the most effort (need data copy).
 - Tagged Functor/Kernel | Most convenient (for us). Whole class becomes a functor.

Lambda Example

- Best for **simple/basic kernels**, but some gotchas with capture of *this* pointer; need to take explicit local copies of class members (e.g. **a** and **x** here).
- **View handles** are **copied by value**; underlying data is shared, not copied.
- Captures scalars, dimensions, and metadata by value.
- Already local variables are fine to use implicitly. (e.g. **y** here).

```
/* Regular C++: x and y are arrays */

for (int i = 0; i < N; ++i) {
    y[i] += a * x[i];
}

/* Kokkos: x and y are Views; a, x are class members; y is local */

const auto a_local = a;
const auto x_local = x;

Kokkos::parallel_for("axy_lambda", N,
    KOKKOS_LAMBDA(const int i) {
        y(i) += a_local * x_local(i);
    }
);
```

Functor Example

- Useful for **reusable kernels** with **explicitly defined inputs**.
- Best (for us) when dealing with data across **multiple classes**.
- Have to **explicitly store the needed scalars and View handles** in the functor.
- Everything is explicit, regardless if it is already local or not.
(e.g. **a**, **x**, and **y** here).

```

/* Functor stores everything the kernel needs */

struct AxyFunctor {
    double a;
    Kokkos::View<double*> x, y;

    // Copy scalar and View handles into the functor
    AxyFunctor(double a_in,
               Kokkos::View<double*> x_in,
               Kokkos::View<double*> y_in)
        : a(a_in), x(x_in), y(y_in) {}

    KOKKOS_INLINE_FUNCTION
    void operator()(const int i) const {
        y(i) += a * x(i);
    }
};

/* Inside the enclosing class's member function:
   a, x are class members; y is a local View */

AxyFunctor kernel{a, x, y};

Kokkos::parallel_for("axy_functor", N, kernel);

```

Tagged Kernel Example

- Useful for **grouping several operations** that **share data in one class**.
- Best (for us) when dealing with data **within a given class**.
- **One functor** can store data for **multiple tagged kernels**.
(e.g. ***TagAdd*** and ***TagScale*** here)
- The **tag** tells Kokkos which `operator()` overload to run at compile time. One class can hold several operations sharing the same data.

```

struct AxyClass {
    struct TagAdd {};
    struct TagScale {};
    double a;
    Kokkos::View<double*> x, y;

    KOKKOS_INLINE_FUNCTION
    void operator()(TagAdd, const int i) const {
        y(i) += x(i); }

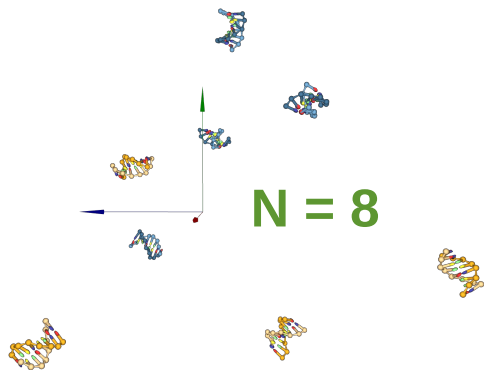
    KOKKOS_INLINE_FUNCTION
    void operator()(TagScale, const int i) const {
        y(i) *= a; }
};

AxyClass op{a, x, y};
Kokkos::parallel_for("axy_tag_add",
    Kokkos::RangePolicy<AxyClass::TagAdd>
        (0, N), op);
Kokkos::parallel_for("axy_tag_scale",
    Kokkos::RangePolicy<AxyClass::TagScale>
        (0, N), op);
  
```

Our Benchmark Suites

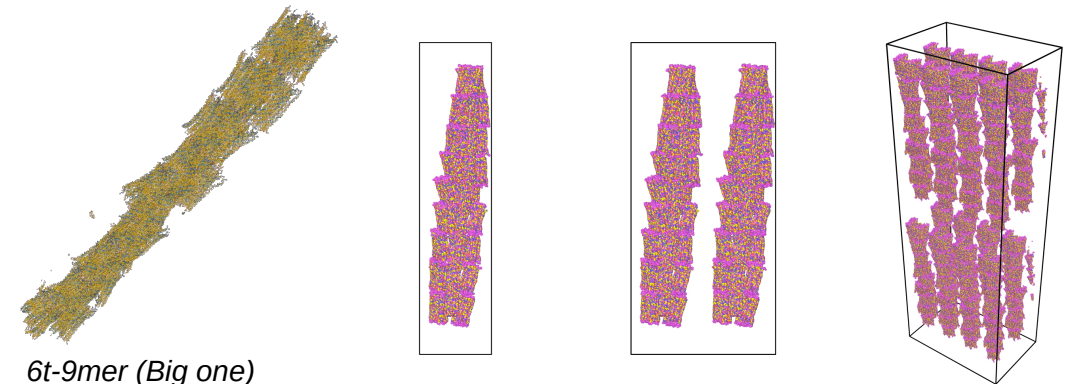
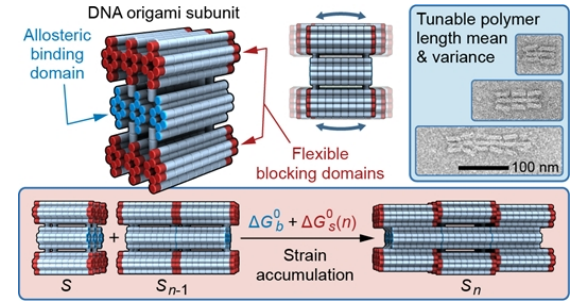
Oligomer |

- Formed of 8bp/16nucl dsDNA strands
- Homogenesis / Load Balanced
- Low Density System
- #Oligomers 'N' = 8, 64, ..., 262K
- Constant Density Throughout 'N'



Polybrick |

- Example dsDNA Nanostructure
- 68kbp/136knucl High Density System
- 1, 2, 4, 8, 16x Replicated



Berengut, Jonathan F., et al. "Self-limiting polymerization of DNA origami subunits with strain accumulation." *ACS nano* 14.12 (2020): 17428-17441.
<https://doi.org/10.1021/acsnano.0c07696>

oxDNA3 - Oligomer

Kokkos @ N >= 4K |

4090 vs 7900 : ~1.8x

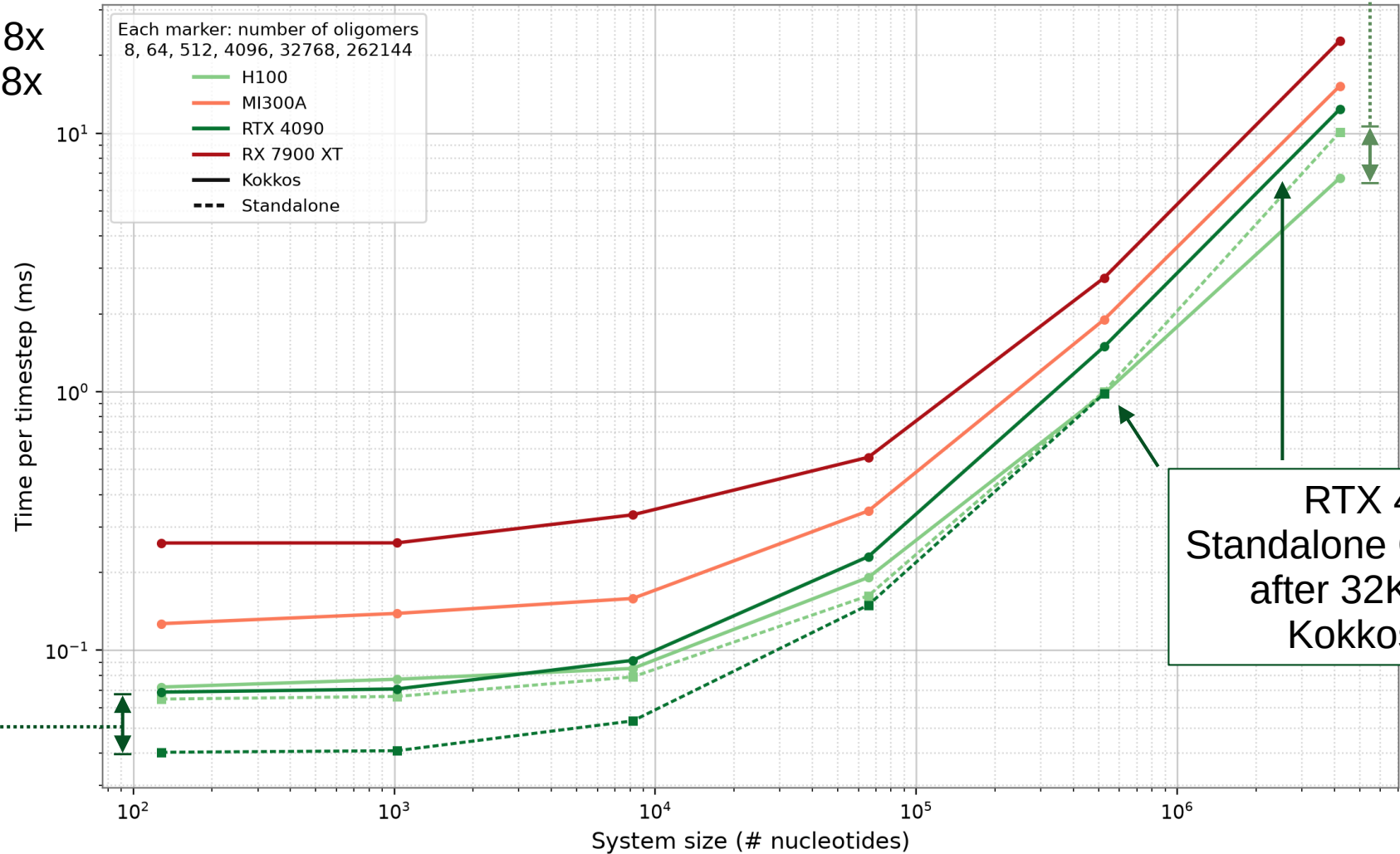
H100 vs 300A : ~1.8x

RTX 4090 |
Standalone faster
than Kokkos.

~1.7x @ N = 8

~1.5x @ N = 32K

Oligomer Scaling Benchmark - Time per Timestep vs System Size
(Lower is Better) - oxDNA3



oxDNA3 - Polybrick

Kokkos @ 136K |

4090 vs 7900 : ~1.9x

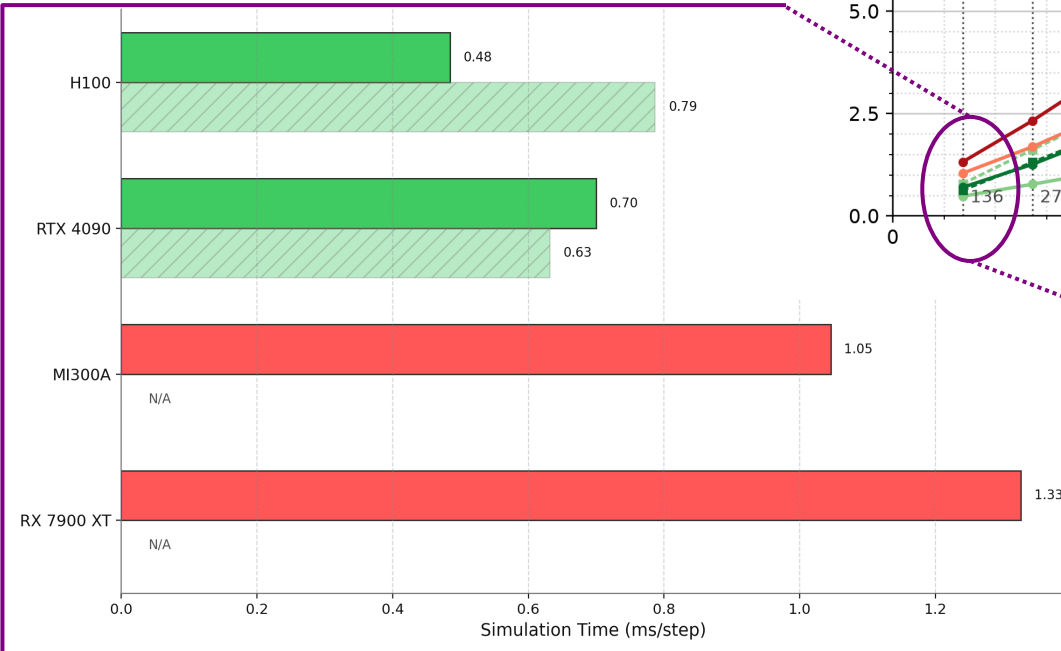
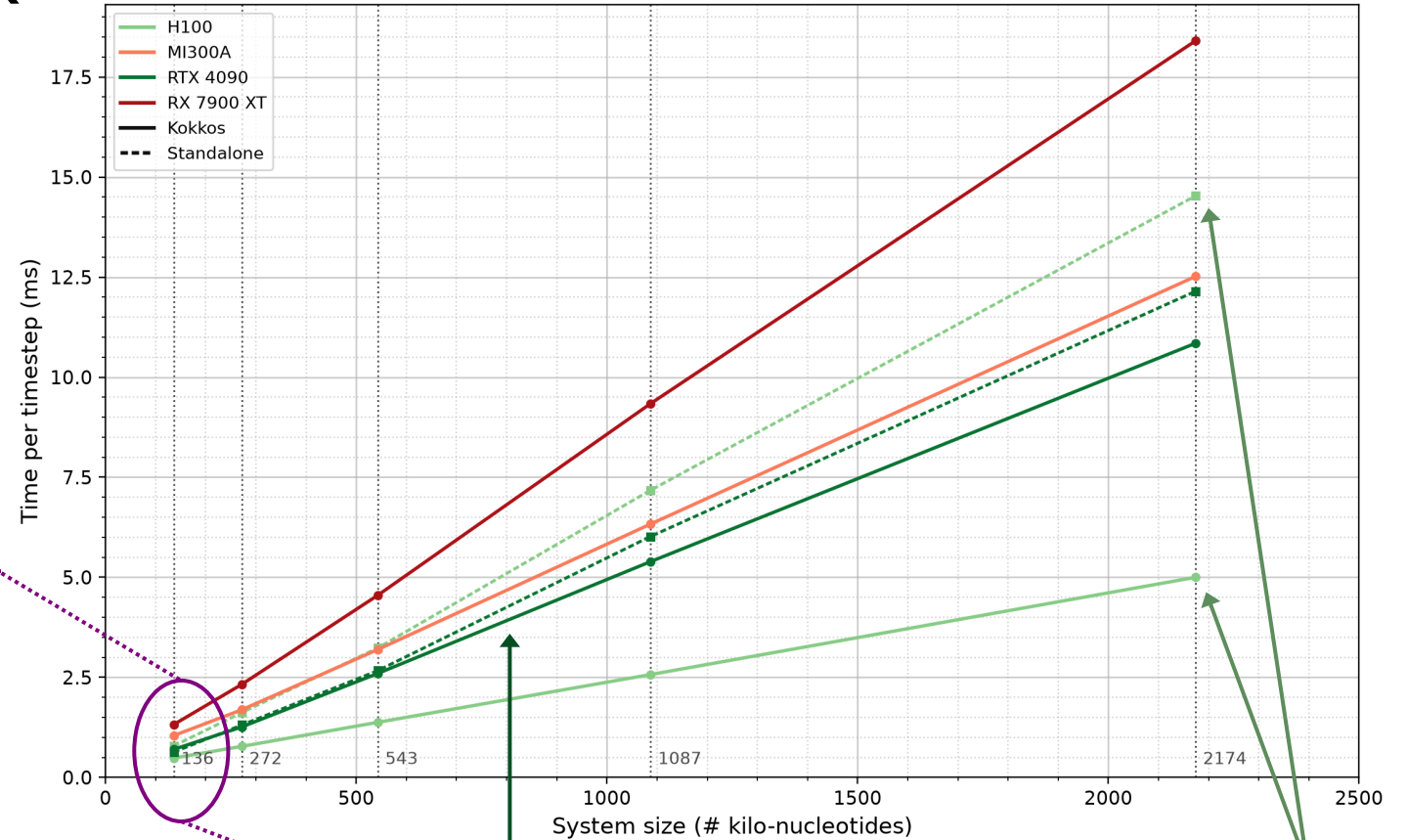
H100 vs 300A : ~2.2x

Kokkos @ 2174K |

4090 vs 7900 : ~1.7x

H100 vs 300A : ~3.0x

Polybrick Scaling Benchmark - Time per Timestep vs System Size
(Lower is Better) - oxDNA3



RTX 4090 |
Kokkos vs Standalone
are very comparable.
~10%

H100 |
Kokkos *much faster*
than Standalone
~1.6-2.9x

oxDNA3 - Salt Concentrations

Standalone |

- One master neighbour list.
- One kernel for bonded.
- One kernel for nonbonded.

Kokkos |

- Kernel for each interaction.
- Trimmed neighbour lists.

$\geq 0.5M$

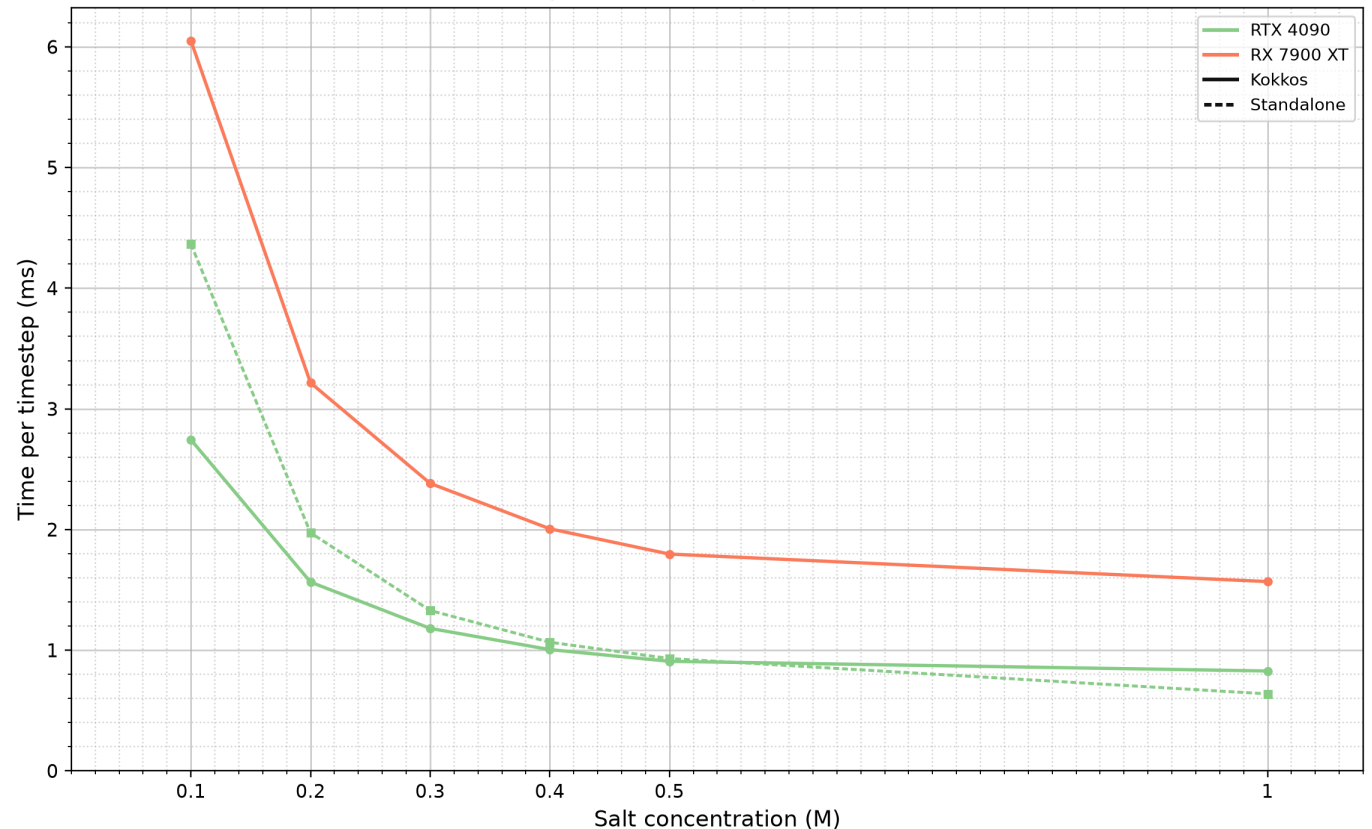
- oxDNA cutoffs $\sim$ to Debye-Hückel Screening

$< 0.5M$

- DH far exceeds oxDNA

Result: Kokkos better suited to Low Salts vs Standalone.

Polybrick Salt Concentration Benchmark - Time per Timestep vs Salt Concentration
(Lower is Better) - oxDNA3



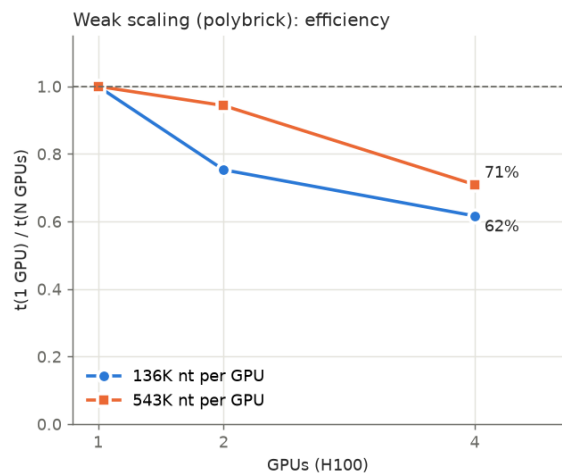
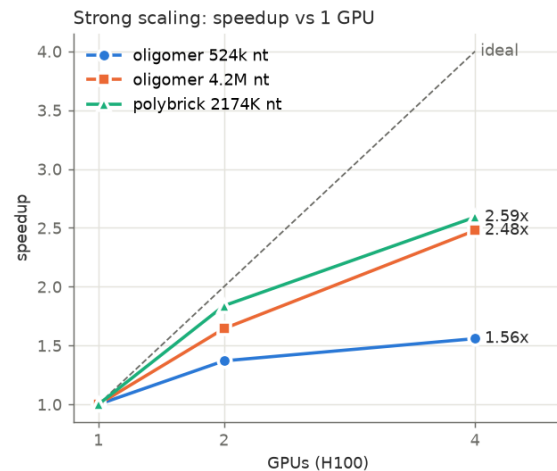
Lower Salt,
Larger Neighbour Lists

Higher Salt,
Smaller Neighbour Lists

Multi-GPU Scaling

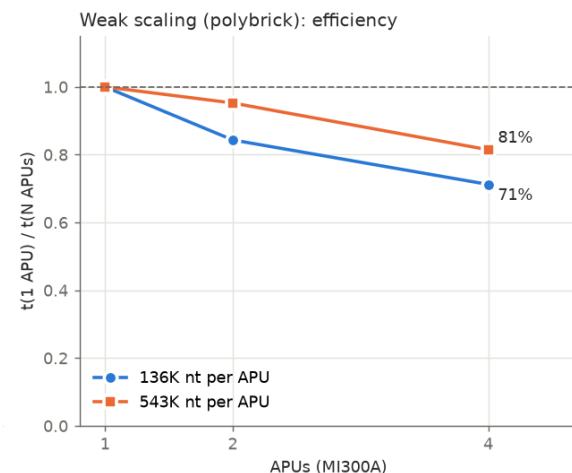
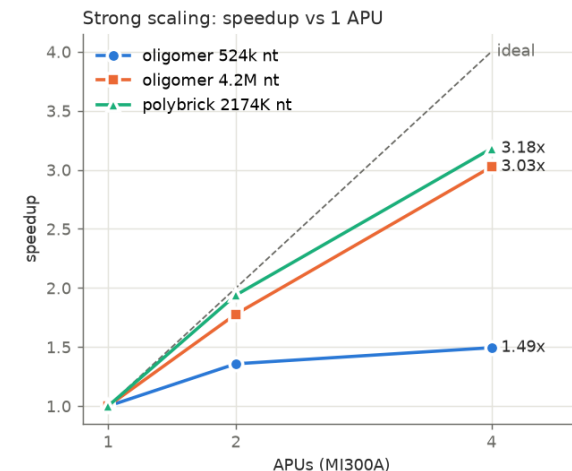
H100 |

1 MPI rank per GPU, CUDA-Aware MPI



MI300A |

1 MPI rank per APU, ROCm-Aware MPI



My Take Homes and Final Comments....

- Kokkos 'problems' are mostly GPU problems – Kokkos is friendly via abstraction
- Computation, not communication by itself – still need some mechanism for inter-GPU communication (GPU-Aware MPI)
- Reliance on **vendor-locked libraries** will still leave you **vendor-locked**
- Kernels *can be* bespoke – Don't take best-practice defaults as gospel

Acknowledgments and Links

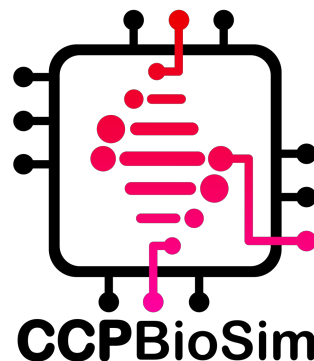
Stan Moore (SNL) |

- One of the Core LAMMPS Developers
- Saved us with some ***nasty*** bugs

*“Make it work,
Make it right,
Make it fast.”*



-
- CCPBioSim | <https://www.ccpbiosim.org/>
 - Digital Research Infrastructure for Integrative Molecular Biology | <https://ccpn.ac.uk/outreach/driimb/>





University of **Strathclyde** Science