IPCC @ RWTH Aachen University

Optimization of multibody and long-range solvers in LAMMPS

Paolo Bientinesi Rodrigo Canales
Markus Höhnerbach Ahmed E. Ismail

Key results – First year
IPCC EMEA meeting Ostrava

Team

RWTH

Prof. Paolo Bientinesi

Rodrigo Canales

Markus Höhnerbach

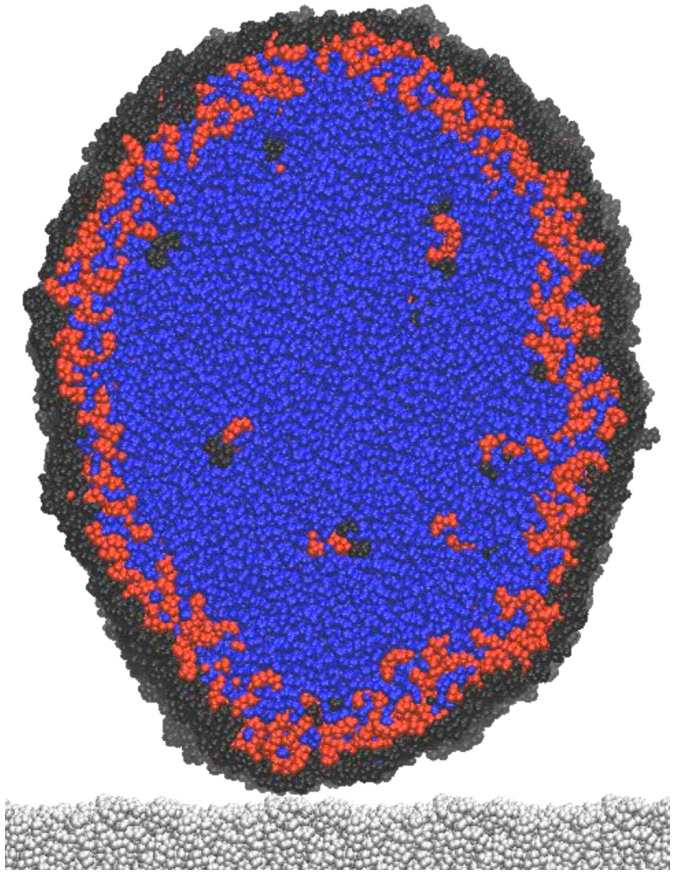
Prof. Ahmed Ismail

Intel

Georg Zitzlsberger Klaus-Dieter Örtel Michael W. Brown

LAMMPS

Large-scale **A**tomic–**M**olecular **M**assively **P**arallel **S**imulator



- Sandia National Labs
  `http://lammps.sandia.gov`
- Wide collection of potentials
- Open source

Parallel Packages for LAMMPS

- LAMMPS parallelization: MPI
- Additional acceleration: OpenMP, GPUs, Intel

Intel Package

- Developed by Michael Brown (Intel)
- Targets Intel's hardware
- Offloading, Vectorization, Precision
- For several potentials

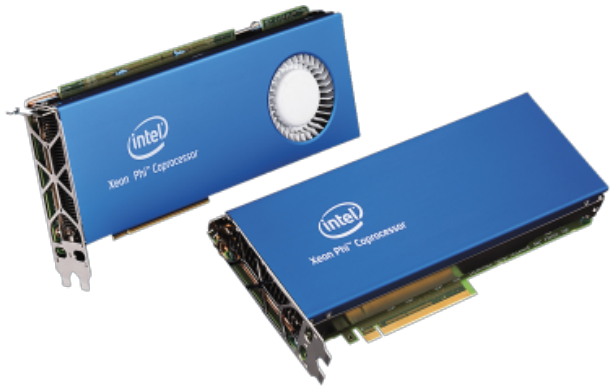


Figure : Xeon Phi coprocessors

Goals

- Optimize core kernels within LAMMPS
  - Multi-threading and **vectorization**
  - Intel Xeon Phi
- **Buckingham** potential, PPPM solver
- **Tersoff** potential, AIREBO potential

Buckingham Potential Optimization

Rodrigo Canales

Pair Potentials

Lennard Jones *(Intel package)*

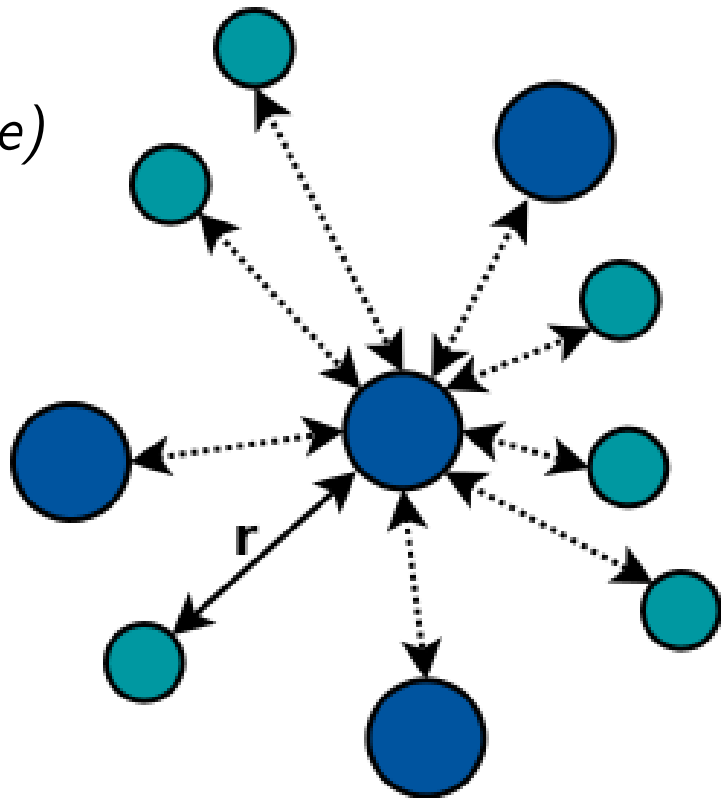
$$\Phi_{lj} = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

Buckingham

$$\Phi_{buck} = Ae^{-r/\rho} - \frac{C}{r^6}$$

$$\Phi_{buck/coul} = \Phi_{buck} + \frac{Cq_i q_j}{\epsilon\, r_{ij}}$$

$$\Phi_{buck/coul/long} = \Phi_{buck} + \frac{Cq_i q_j}{\epsilon\, r_{ij}} \mathrm{erfc}(\alpha r_{ij})$$



Buck Potential Optimization

- USER-INTEL package as base of the development
- Data Packing for parameters
- Alignment of force and position arrays
- Multiple precision support
- Enable Xeon Phi Offloading
- **Vectorization: Pragma SIMD**

Speedup Xeon (single-threaded)

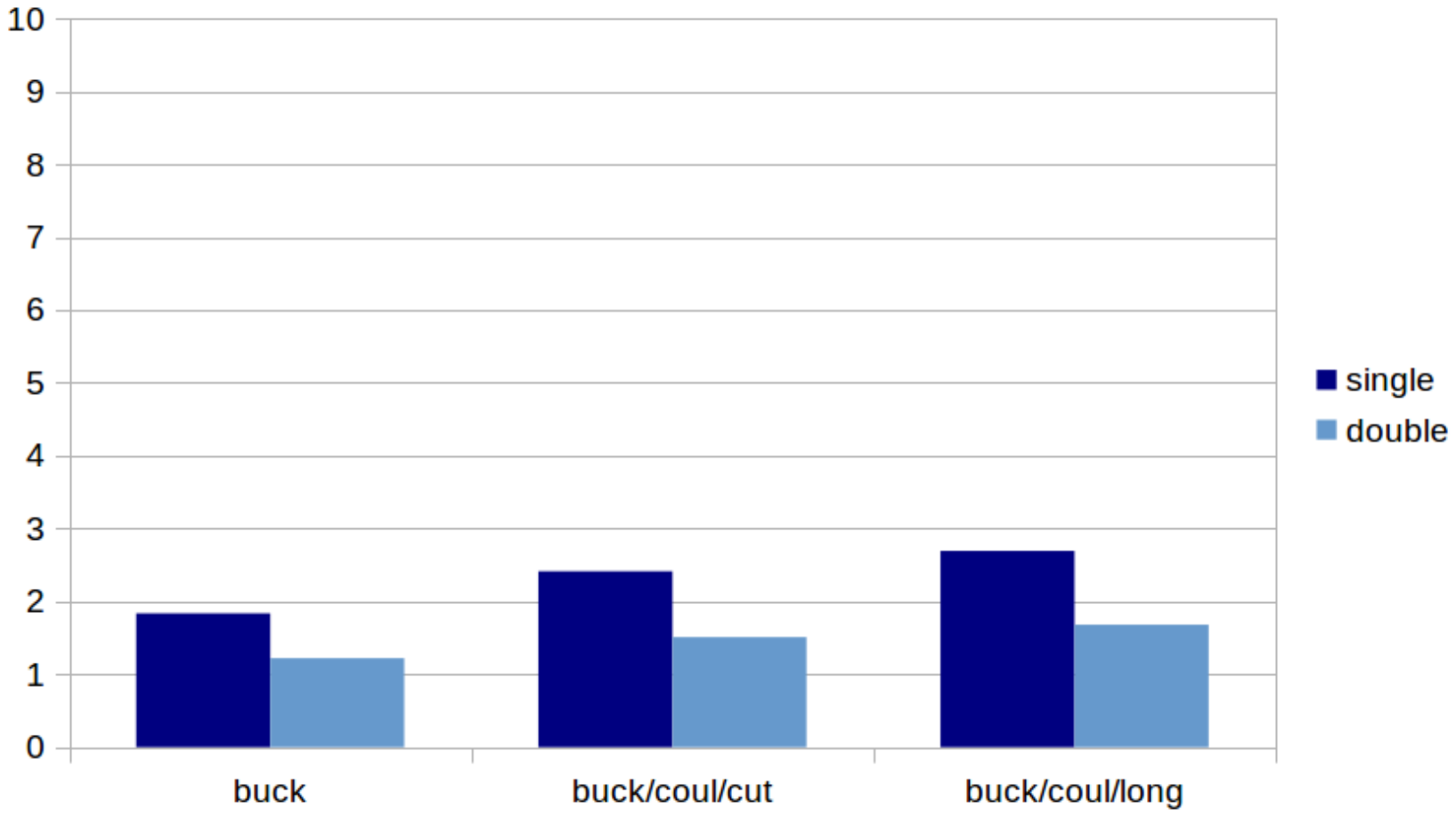


Figure : Speedup on the Xeon E5-2650 (Sandy Bridge)

Speedup Xeon Phi (single-threaded, native)

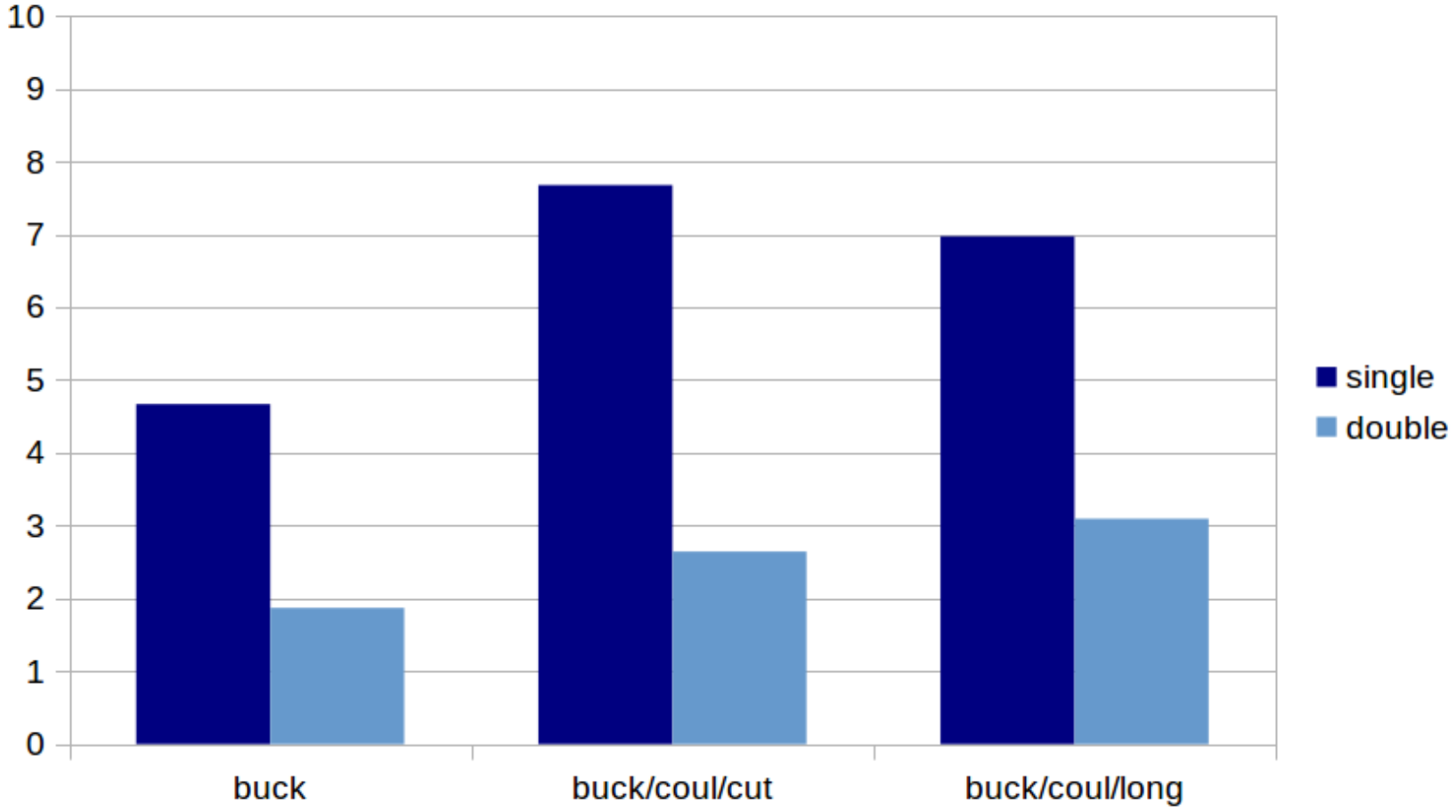


Figure : Speedup on Xeon Phi 5110P

KNC Intrinsics Vectorization

- Gather operations in neighbor loading
- Replace by in-register transpose 4x8 or 4x16

Templating intrinsics:	760 lines
Implementation pair/buck:	330 lines

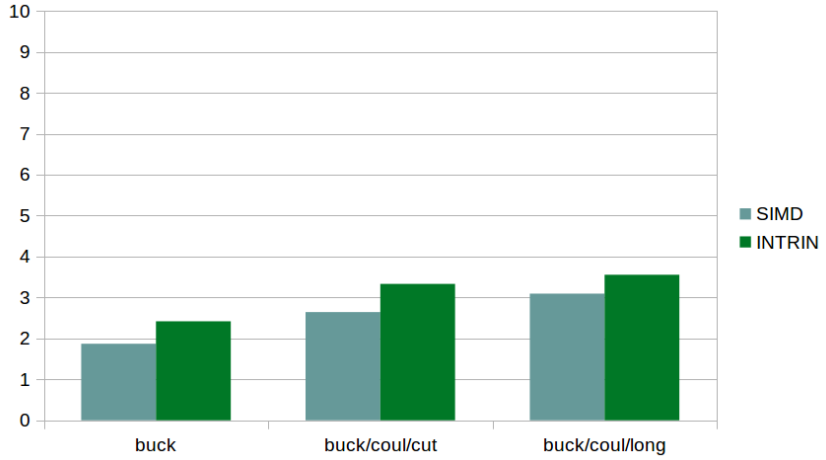


Figure : Speedup comparison on the Xeon Phi (Double)

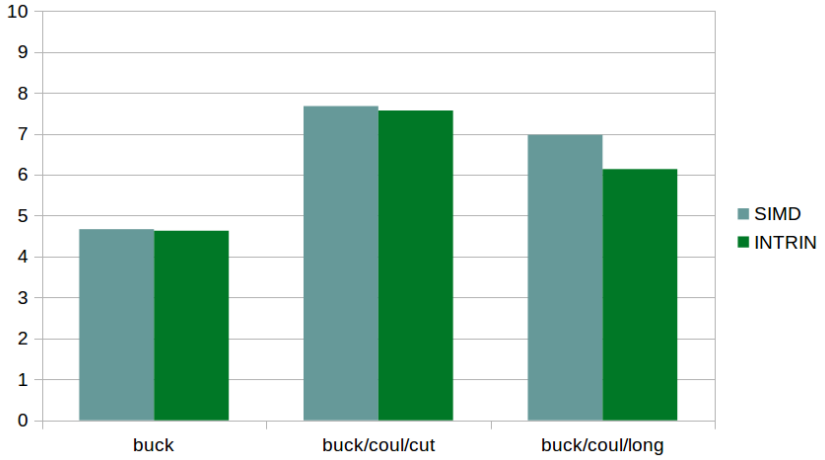


Figure : Speedup comparison on the Xeon Phi (Single)

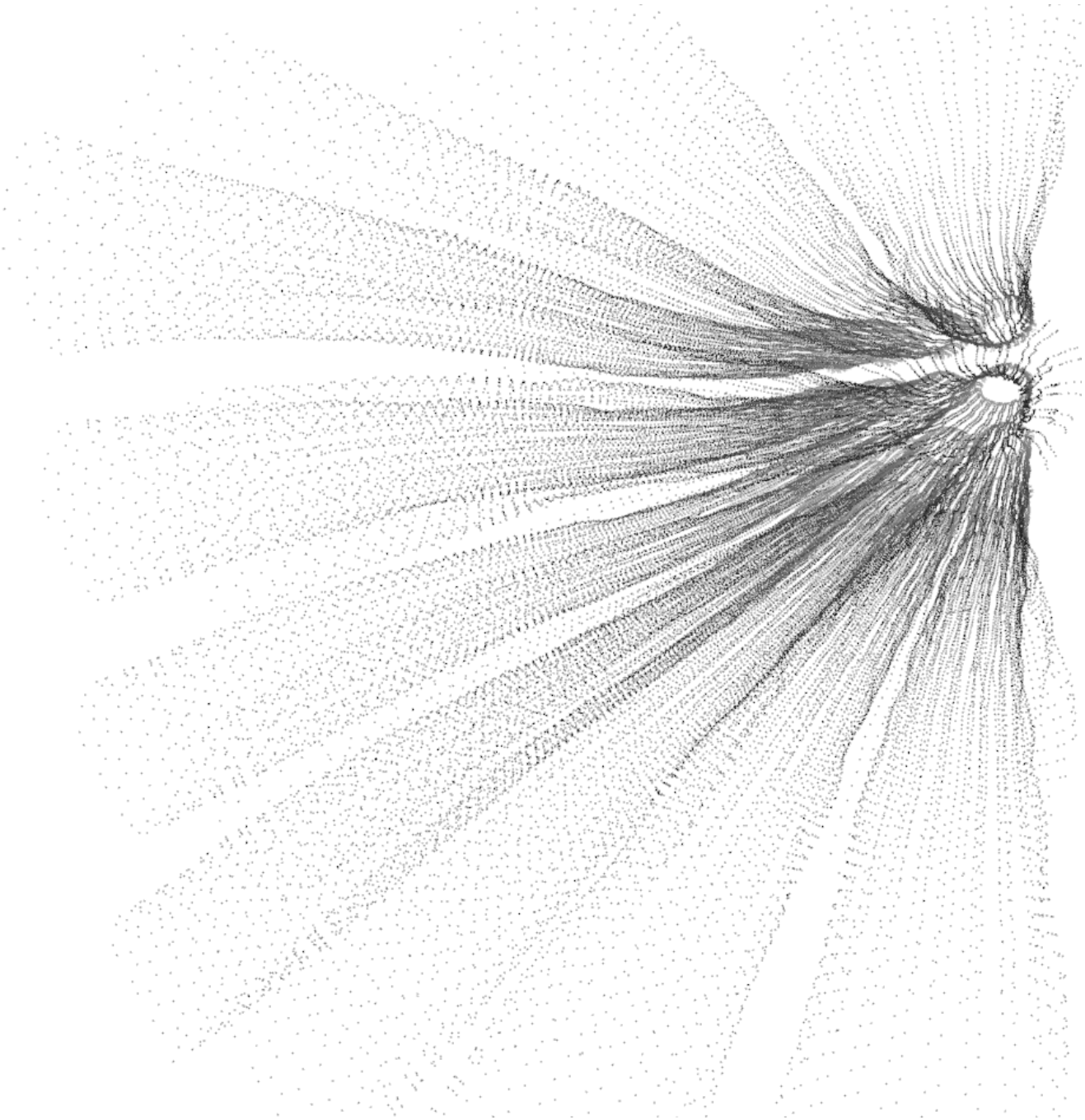
Runtime on Full System

576'000 atoms, double precision

Xeon Phi Native	Base	253s
(240 Threads)	SIMD	202s
Xeon ($\times 2$) + Xeon Phi	Base	120s
(32 + 240 Threads)	SIMD	77s

Multibody potentials
Modernization of the Tersoff potential

Markus Höhnerbach



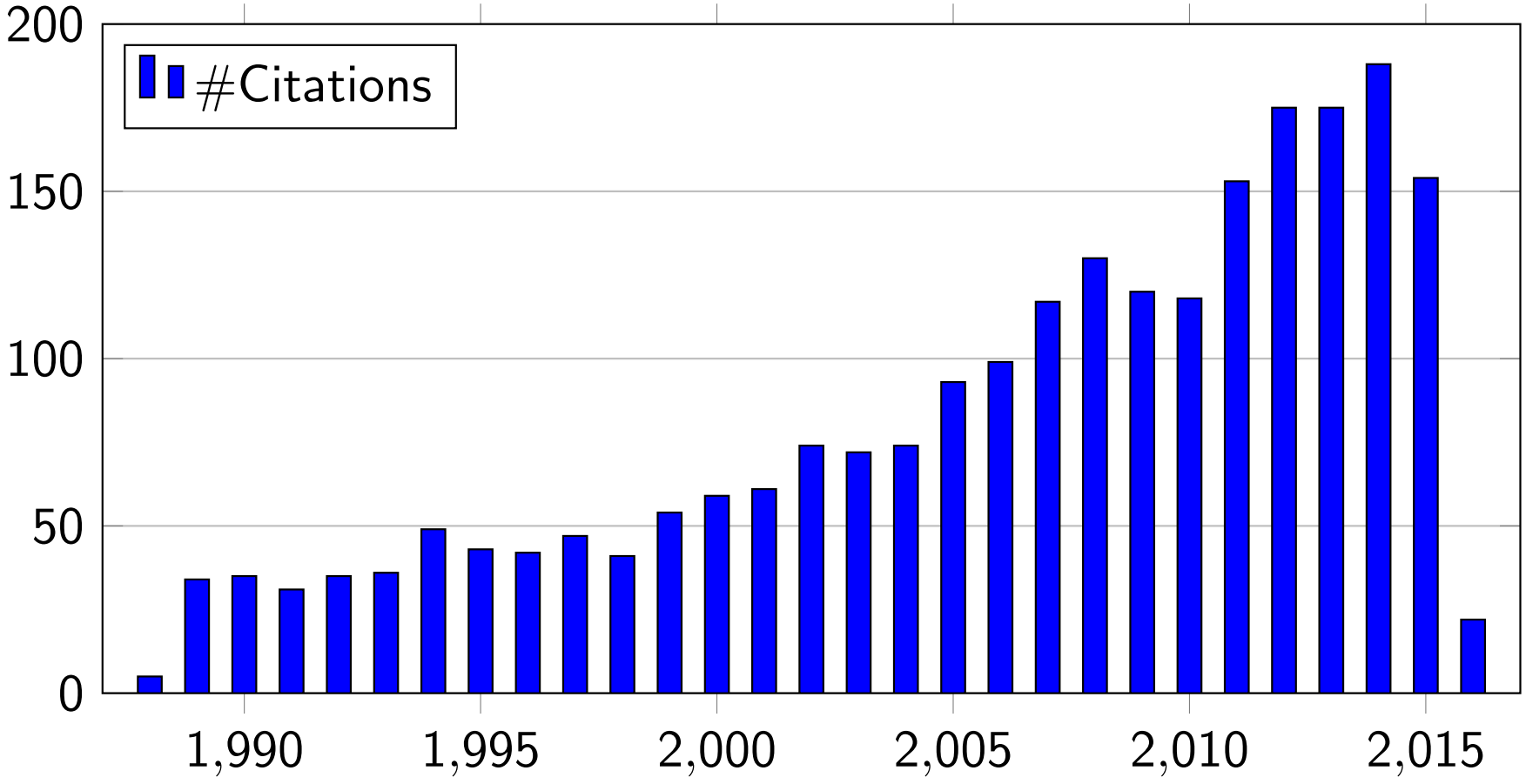
The Tersoff potential

$$V = \sum_i \sum_{j \in \mathcal{N}_i} \overbrace{f_C(r_{ij}) \left[f_R(r_{ij}) + \mathbf{b_{ij}} f_A(r_{ij}) \right]}^{V(i,j,\zeta_{ij})} \tag{1}$$

$$b_{ij} = (1 + \beta^\eta \zeta_{ij}^\eta)^{-\frac{1}{2\eta}} \tag{2}$$

$$\zeta_{ij} = \sum_{k \in \mathcal{N}_i \setminus \{j\}} \underbrace{f_C(r_{ik}) g(\theta_{ijk}) \exp(\lambda_3 (r_{ij} - r_{ik}))}_{\zeta(i,j,k)} \tag{3}$$

Popularity

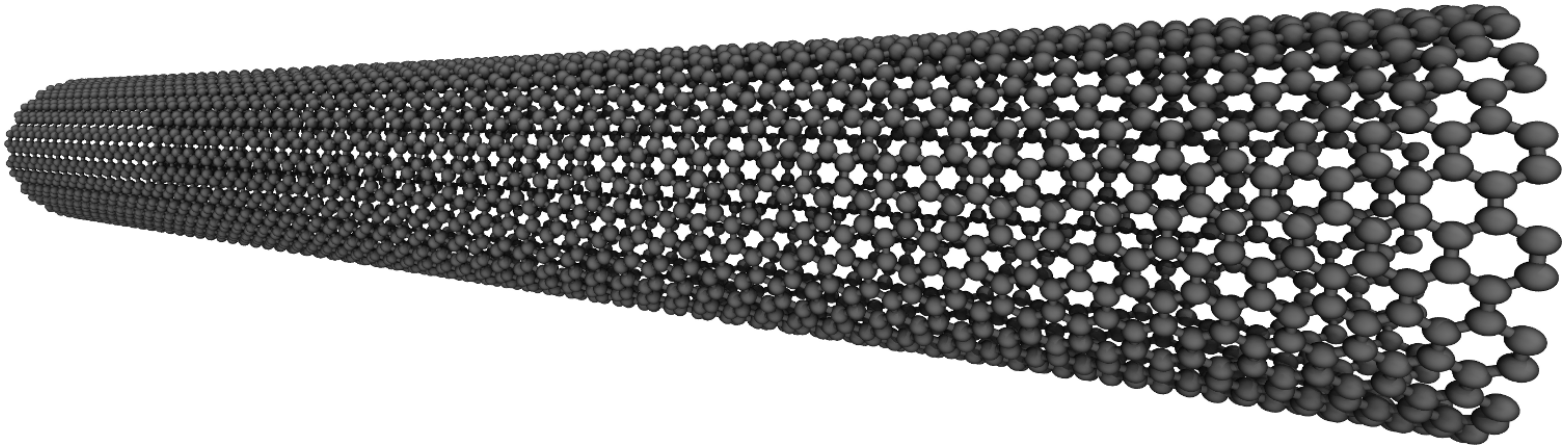


- Tersoff potential: Widely used, fairly simple (~700 LOC)
- Previous work for GPU: EAM[a], Stillinger-Weber[b] and Tersoff [c]

The Tersoff Algorithm

for *i in local atoms of the current thread* **do**
  for *j in atoms neighboring i* **do**
    $\zeta_{ij} \leftarrow 0;$
    for *k in atoms neighboring i* **do**
      $\zeta_{ij} \leftarrow \zeta_{ij} + \zeta(i,j,k);$
    $E \leftarrow E + V(i,j,\zeta_{ij});$
    $F_i \leftarrow F_i - \partial_{x_i} V(i,j,\zeta_{ij});$
    $F_j \leftarrow F_j - \partial_{x_j} V(i,j,\zeta_{ij});$
    $\delta\zeta \leftarrow \partial_{\zeta} V(i,j,\zeta_{ij});$
    for *k in atoms neighboring i* **do**
      $F_i \leftarrow F_i - \delta\zeta \cdot \partial_{x_i}\zeta(i,j,k);$
      $F_j \leftarrow F_j - \delta\zeta \cdot \partial_{x_j}\zeta(i,j,k);$
      $F_k \leftarrow F_k - \delta\zeta \cdot \partial_{x_k}\zeta(i,j,k)$

Close-Up



Challenges

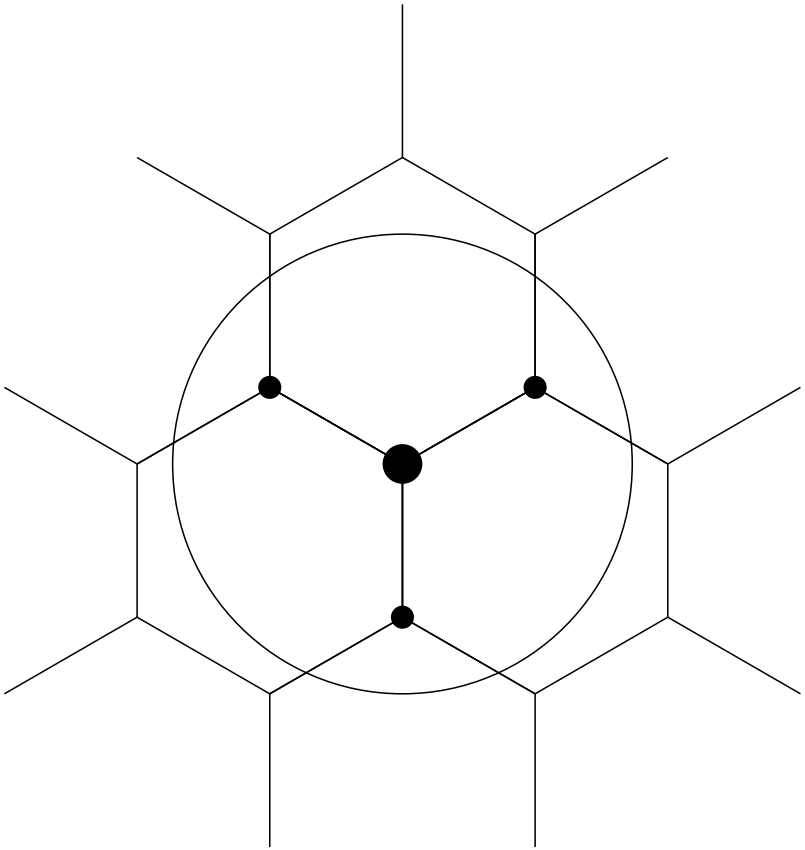


Figure : Graphene

- Few neighbors
- Fewer interactions

Vectorization

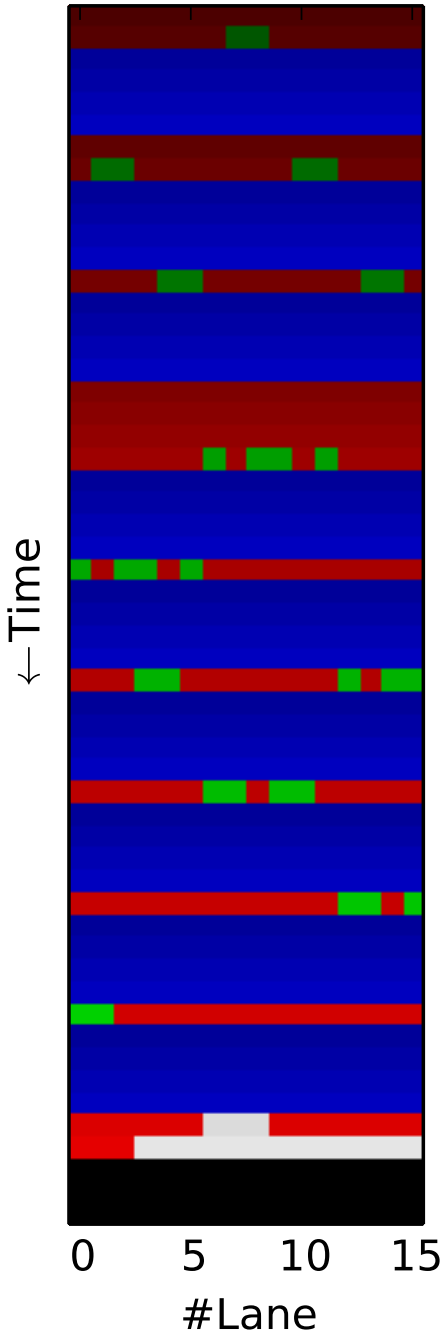
"J" algorithm

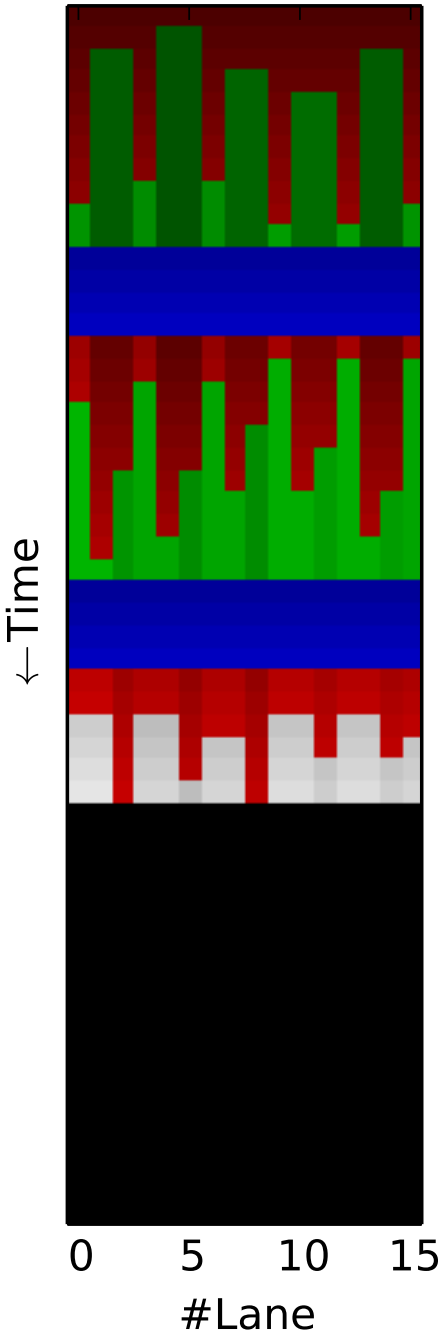
```
for i do
    for j ∈ N_i do
        skip cutoff;
        ...;
        for k ∈ N_i \ {j} do
            skip cutoff;
            ...;
        ...;
        for k ∈ N_i \ {j} do
            skip cutoff;
            ...;
```

"I" algorithm

```
for i do
    for j ∈ N_i do
        skip cutoff;
        ...;
        for k ∈ N_i \ {j} do
            skip cutoff;
            ...;
        ...;
        for k ∈ N_i \ {j} do
            skip cutoff;
            ...;
```

K Loop





Abstraction

```
typedef vector_routines<double, double, AVX> v;
typedef v::fvec fvec;
fvec a(1);
fvec b(2);
fvec c = v::recip(a + b);
```

Features

- Supports single, double and mixed precision
- Supports scalar, SSE4.2, AVX, AVX2, IMCI, AVX-512, array notation (Cilk)

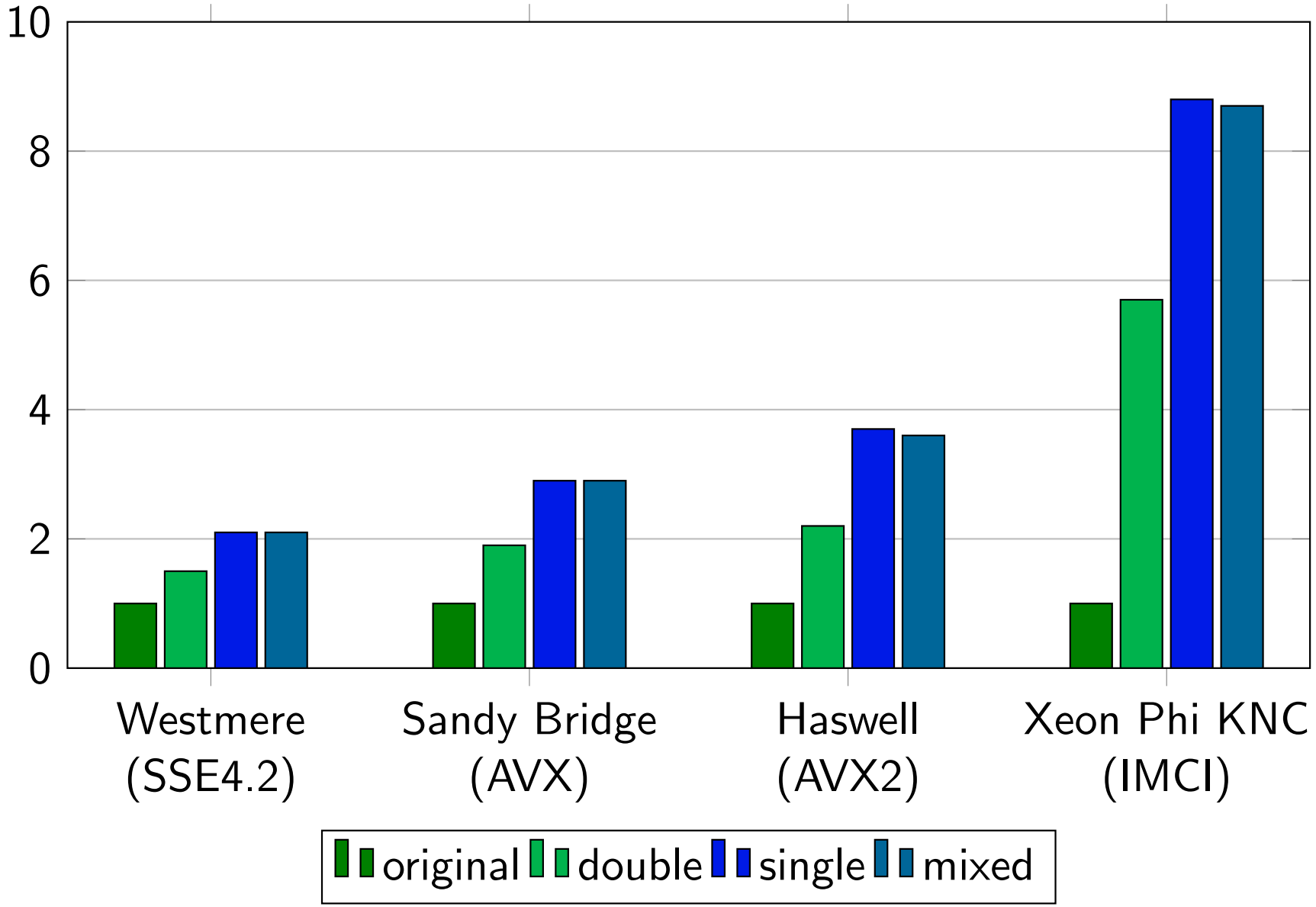
Advantages

- Maintainability
- Testing (through AN)
- Portability
- Thin wrapper

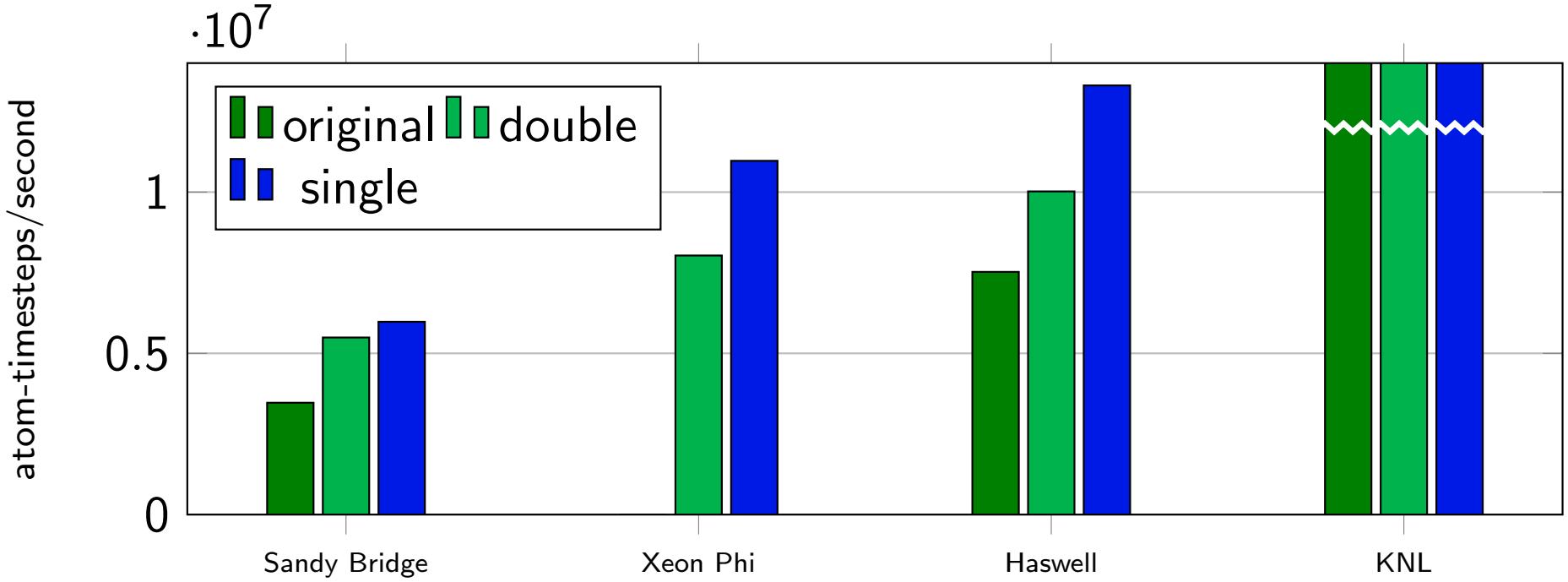
KNL Readiness

- Intrinsics abstraction already supports AVX-512
- Compilation possible for `-xMIC-AVX512`
- Running under Intel SDE `sde -knl -- ...`
- Has been tested on KNL prototypes by Intel employees
- We already have benchmarks prepared for the point when performance data can be shared

Portable Optimization (single-threaded, native)



Impact on a Realistic Simulation (multi-threaded)



Configuration

Arch.	Model	Year	Cores
Haswell	2x Xeon E5-2680 v3	2014	24
Sandy Bridge	2x Xeon E5-2450	2012	16
	1x Xeon Phi 5110P	2012	60

Dissemination

Oct'15	Code dungeon EMEA IPCC meeting, Munich	✓
Nov'15	`github.com/HPAC` Code, tests and benchmarks	 *in progress*
Nov'15	Talk + paper at SC'15 Workshop	✓
Dec'15	Code integrated into LAMMPS	✓
Dec'15	IXPUG: Vectorization WG	*in progress*

Future Work

PPPM Long-ranged solver, used in almost any simulation

- Special focus on vectorization
- Particle-to-grid and grid-to-particle step

AIREBO Complex potential for simulation of hydrocarbons

- Some reuse from Tersoff optimization
- Challenging vectorization: Searches
- Challenging vectorization: Data-dependent, unlikely branches
- Challenging vectorization: Deep loop-nests with low trip counts

intel®