High-Performance & Automatic Computing

Fast & portable code for complex molecular dynamics simulations

Paolo Bientinesi & Markus Höhnerbach

Aachen Institute for Computational Engineering Science

RWTH Aachen University

November 9, 2018

ICES, University of Texas at Austin

Deutsche Forschungsgemeinschaft DFG

HPAC High Performance and Automatic Computing

RWTH AACHEN UNIVERSITY

The world of scientific computing

The world of scientific computing

$$\boxed{\boldsymbol{y} = X\boldsymbol{\beta} + Z\boldsymbol{u} + \boldsymbol{\epsilon}}$$

$$\boxed{\min_x \|A\mathbf{x} - \mathbf{b}\|^2 + \|\Gamma\mathbf{x}\|^2}$$

Linear mixed models

$$\boxed{V_{LJ} = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]}$$

Lennard-Jones potential

$$\boxed{i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) = \left[\frac{-2\hbar}{2\mu}\nabla^2 + V(\mathbf{r},t)\right]\Psi(\mathbf{r},t)}$$

Schrödinger eqn.

$$\vdots$$

The world of scientific computing

$$\boxed{\boldsymbol{y} = X\beta + Z\boldsymbol{u} + \epsilon}$$

$$\boxed{\min_x \|A\mathbf{x} - \mathbf{b}\|^2 + \|\Gamma\mathbf{x}\|^2}$$

Linear mixed models

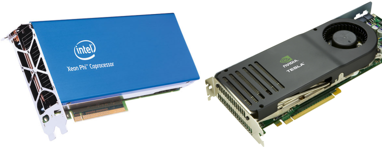
$$\boxed{V_{LJ} = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]}$$

Lennard-Jones potential

$$\boxed{i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) = \left[\frac{-2\hbar}{2\mu}\nabla^2 + V(\mathbf{r},t)\right]\Psi(\mathbf{r},t)}$$

Schrödinger eqn.

$$\vdots$$



The world of scientific computing

$$\boxed{\boldsymbol{y} = X\beta + Z\boldsymbol{u} + \boldsymbol{\epsilon}}$$

$$\boxed{\min_x \|A\mathbf{x} - \mathbf{b}\|^2 + \|\Gamma\mathbf{x}\|^2}$$

Linear mixed models

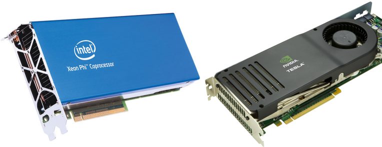
$$\boxed{V_{LJ} = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]}$$

Lennard-Jones potential

$$\boxed{i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) = \left[\frac{-2\hbar}{2\mu}\nabla^2 + V(\mathbf{r},t)\right]\Psi(\mathbf{r},t)}$$

Schrödinger eqn.

$$\vdots$$



The world of scientific computing

$$\boxed{\boldsymbol{y} = X\beta + Z\boldsymbol{u} + \boldsymbol{\epsilon}}$$

$$\boxed{\min_x \|A\mathbf{x} - \mathbf{b}\|^2 + \|\Gamma\mathbf{x}\|^2}$$

Linear mixed models

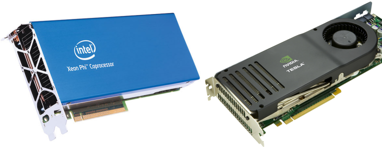
$$\boxed{V_{LJ} = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]}$$

Lennard-Jones potential

$$\boxed{i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) = \left[\frac{-2\hbar}{2\mu}\nabla^2 + V(\mathbf{r},t)\right]\Psi(\mathbf{r},t)}$$

Schrödinger eqn.

$\vdots$



1) Linear Algebra, Applications

2) Tensor Operations

3) Molecular Dynamics

Linear Algebra

Generalized Least Squares	$b := (X^T M^{-1} X)^{-1} X^T M^{-1} y$	$n > m$; $M \in \mathbb{R}^{n \times n}$, SPD; $X \in \mathbb{R}^{n \times m}$; $y \in \mathbb{R}^{n \times 1}$
Signal Processing	$x := \left(A^{-T} B^T B A^{-1} + R^T L R\right)^{-1} A^{-T} B^T B A^{-1} y$	
Kalman Filter	$K_k := P_k^b H^T (H P_k^b H^T + R)^{-1};\; x_k^a := x_k^b + K_k(z_k - H x_k^b);\; P_k^a := (I - K_K H)\, P_k^b$	
Ensemble Kalman Filter	$X^a := X^b + \left(B^{-1} + H^T R^{-1} H\right)^{-1} \left(Y - H X^b\right)$	
Image Restoration	$x_k := (H^T H + \lambda \sigma^2 I_n)^{-1} (H^T y + \lambda \sigma^2 (v_{k-1} - u_{k-1}))$	
Rand. Matrix Inversion	$X_{k+1} := S(S^T A S)^{-1} S^T + (I_n - S(S^T A S)^{-1} S^T A) X_k (I_n - A S (S^T A S)^{-1} S^T)$	
Stochastic Newton	$B_k := \frac{k}{k-1} B_{k-1}(I_n - A^T W_k((k-1) I_l + W_k^T A B_{k-1} A^T W_k)^{-1} W_k^T A B_{k-1})$	
Optimization	$x_f := W A^T (A W A^T)^{-1} (b - Ax);\quad x_o := W(A^T (A W A^T)^{-1} A x - c)$	
Tikhonov Regularization	$x := (A^T A + \Gamma^T \Gamma)^{-1} A^T b$	$A \in \mathbb{R}^{n \times m}$; $\Gamma \in \mathbb{R}^{m \times m}$; $b \in \mathbb{R}^{n \times 1}$
Gen. Tikhonov Reg.	$x := (A^T P A + Q)^{-1} (A^T P b + Q x_0)$	$P \in \mathbb{R}^{n \times n}$, SSPD; $Q \in \mathbb{R}^{m \times m}$, SSPD; $x_0 \in \mathbb{R}^{m \times 1}$
LMMSE estimator	$K_{t+1} := C_t A^T (A C_t A^T + C_z)^{-1};\; x_{t+1} := x_t + K_{t+1}(y - A x_t);\; C_{t+1} := (I - K_{t+1} A) C_t$	

$$x := A(B^TB + A^TR^T\Lambda RA)^{-1}B^TBA^{-1}y$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T(HC_\dagger H^T)^{-1} \end{cases}$$

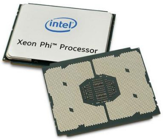
$$E := Q^{-1}U(I + U^TQ^{-1}U)^{-1}U^T$$

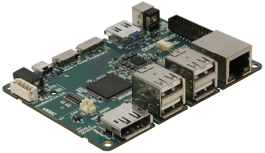
. . .

$$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y$$

$$\begin{cases} C_\dagger := P C P^T + Q \\ K := C_\dagger H^T (H C_\dagger H^T)^{-1} \end{cases}$$

$$E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T \qquad \ldots$$





MUL ADD MOV

MOVAPD

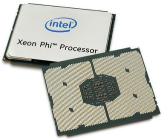
VFMADDPD ...

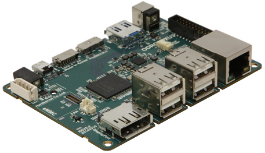
$$x := A(B^TB + A^TR^T\Lambda RA)^{-1}B^TBA^{-1}y$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T(HC_\dagger H^T)^{-1} \end{cases}$$

$$E := Q^{-1}U(I + U^TQ^{-1}U)^{-1}U^T \qquad \ldots$$

↓ ?





MUL ADD MOV

MOVAPD

VFMADDPD ...

$$x := A(B^TB + A^TR^T\Lambda RA)^{-1}B^TBA^{-1}y$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T(HC_\dagger H^T)^{-1} \end{cases}$$

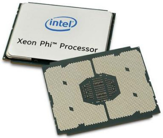
$$E := Q^{-1}U(I + U^TQ^{-1}U)^{-1}U^T$$

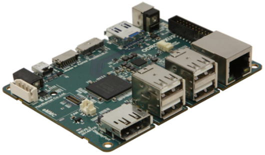
...

$\boxed{y := \alpha x + y}$ $\boxed{LU = A}$ $\cdots$ $\boxed{C := \alpha AB + \beta C}$

$\boxed{X := A^{-1}B}$ $\boxed{C := AB^T + BA^T + C}$ $\boxed{X := L^{-1}ML^{-T}}$ $\boxed{QR = A}$

BLAS ↓ **BLIS** ↓ **LAPACK** ↓ ...





MUL ADD MOV

MOVAPD

VFMADDPD ...

$$x := A(B^TB + A^TR^T\Lambda RA)^{-1}B^TBA^{-1}y$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T(HC_\dagger H^T)^{-1} \end{cases}$$

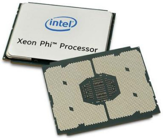
$$E := Q^{-1}U(I + U^TQ^{-1}U)^{-1}U^T \quad \ldots$$

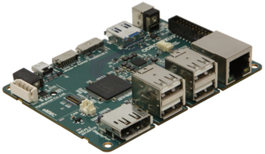
?

$\boxed{y := \alpha x + y}$ $\boxed{LU = A}$ $\cdots$ $\boxed{C := \alpha AB + \beta C}$

$\boxed{X := A^{-1}B}$ $\boxed{C := AB^T + BA^T + C}$ $\boxed{X := L^{-1}ML^{-T}}$ $\boxed{QR = A}$

BLAS **BLIS** **LAPACK** **...**





MUL ADD MOV

MOVAPD

VFMADDPD ...

$$x := A(B^TB + A^TR^T\Lambda RA)^{-1}B^TBA^{-1}y$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T(HC_\dagger H^T)^{-1} \end{cases}$$

$$E := Q^{-1}U(I + U^TQ^{-1}U)^{-1}U^T \qquad \ldots$$

LINEAR ALGEBRA MAPPING PROBLEM (LAMP)

$y := \alpha x + y$

$:= \alpha AB + \beta C$

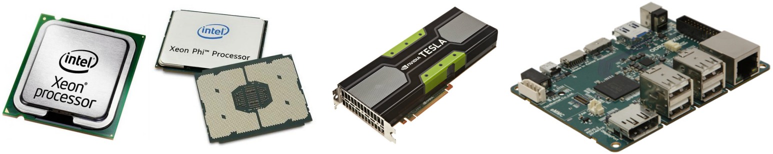
$X := A^{-1}B$

$C := A$

$^{-1}ML^{-T}$

$QR = A$

BLAS **BLIS** **LAPACK** ...



MUL ADD MOV

MOVAPD

VFMADDPD ...

Example

$$w := AB^{-1}c, \quad \text{SPD}(B)$$

H. Barthels
C. Psarras

Naive ← NEVER!!

```
w = A*inv(B)*c
```

Recommended

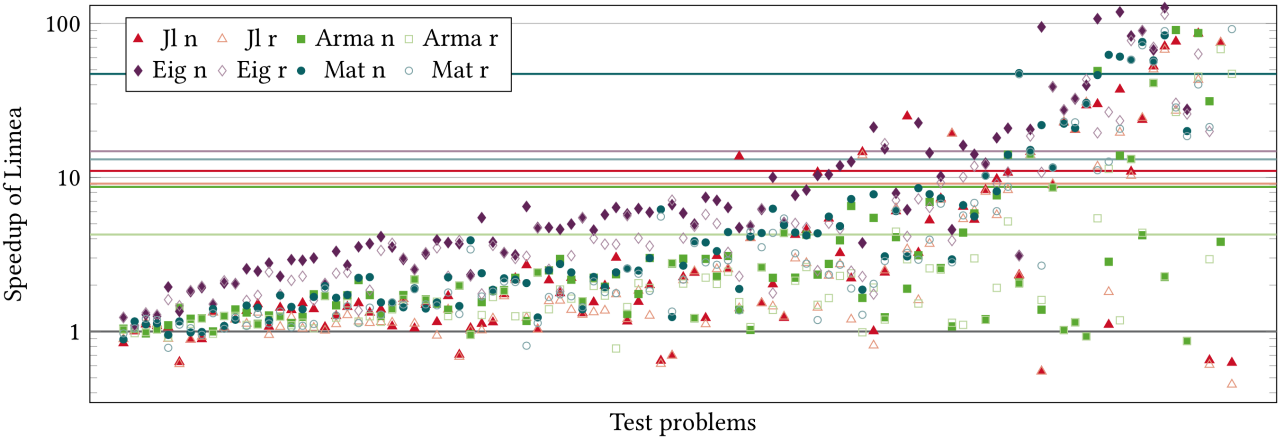
```
w = A*(B\c)
```

Expert

```
L = Chol(B)
w = A * (L'\(L\c))
```

Generated — "Linnea"

```
ml0 = A; ml1 = B; ml2 = c;
potrf!('L', ml1)
trsv!('L', 'N', 'N', ml1, ml2)
trsv!('L', 'T', 'N', ml1, ml2)
ml3 = Array{Float64}(10)
gemv!('N', 1.0, ml0, ml2, 0.0, ml3)
w = ml3
```


100
10
1
Speedup of Linnea
Test problems
▲ Jl n
△ Jl r
■ Arma n
□ Arma r
◆ Eig n
◇ Eig r
● Mat n
○ Mat r

Tensor Operations

Tensor Operations

DFT: Generation of Hamiltonian and Overlap Matrices

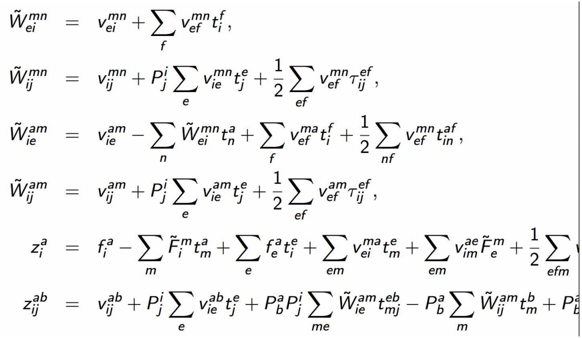
$$(S)_{G',G} = \sum_{a} \sum_{L=(l,m)} \left(A_L^{a,G'}\right)^* A_L^{a,G} + \left(B_L^{a,G'}\right)^* B_L^{a,G} \left\|\dot{u}_{l,a}\right\|^2$$

$$\begin{aligned}(H)_{G',G} = \sum_{a} \sum_{L',L} &\left(A^*_{L',a,t'}\, T^{[AA]}_{L',L;a}\, A_{L,a,t}\right) + \left(A^*_{L',a,t'}\, T^{[AB]}_{L',L;a}\, B_{L,a,t}\right) \\ &+ \left(B^*_{L',a,t'}\, T^{[BA]}_{L',L;a}\, A_{L,a,t}\right) + \left(B^*_{L',a,t'}\, T^{[BB]}_{L',L;a}\, B_{L,a,t}\right)\end{aligned}$$

Tensor Operations

Coupled-Cluster methods

$$
\begin{aligned}
\tau_{ij}^{ab} &= t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b, \\
\tilde{F}_e^m &= f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f, \\
\tilde{F}_e^a &= (1-\delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} v_{ef}^{mn} t_{mn}^{af} + \sum_{fn} v_{ef}^{an} t_n^f, \\
\tilde{F}_i^m &= (1-\delta_{mi}) f_i^m + \sum_e \tilde{F}_e^m t_i^e + \frac{1}{2} \sum_{nef} v_{ef}^{mn} t_{in}^{ef} + \sum_{fn} v_{if}^{mn} t_n^f,
\end{aligned}
$$



$$
\begin{aligned}
\tilde{W}_{ei}^{mn} &= v_{ei}^{mn} + \sum_f v_{ef}^{mn} t_i^f, \\
\tilde{W}_{ij}^{mn} &= v_{ij}^{mn} + P_j^i \sum_e v_{ie}^{mn} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{mn} \tau_{ij}^{ef}, \\
\tilde{W}_{ie}^{am} &= v_{ie}^{am} - \sum_n \tilde{W}_{ei}^{mn} t_n^a + \sum_f v_{ef}^{ma} t_i^f + \frac{1}{2} \sum_{nf} v_{ef}^{mn} t_{in}^{af}, \\
\tilde{W}_{ij}^{am} &= v_{ij}^{am} + P_j^i \sum_e v_{ie}^{am} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{am} \tau_{ij}^{ef}, \\
z_i^a &= f_i^a - \sum_m \tilde{F}_i^m t_m^a + \sum_e f_e^a t_i^e + \sum_{em} v_{ei}^{ma} t_m^e + \sum_{em} v_{im}^{ae} \tilde{F}_e^m + \frac{1}{2} \sum_{efm} \\
z_{ij}^{ab} &= v_{ij}^{ab} + P_j^i \sum_e v_{ie}^{ab} t_j^e + P_b^a P_j^i \sum_{me} \tilde{W}_{ie}^{am} t_{mj}^{eb} - P_b^a \sum_m \tilde{W}_{ij}^{am} t_m^b + P_b^a
\end{aligned}
$$

credits to D. Matthews, E. Solomonik, J. Stanton, and J. Gauss

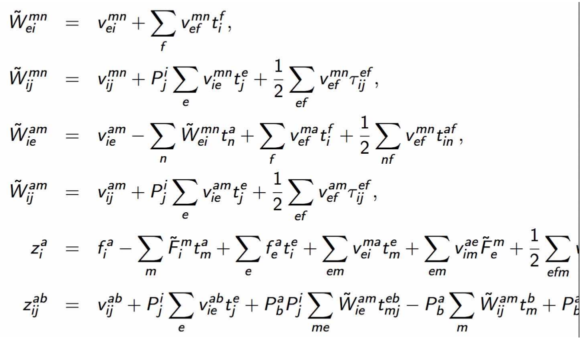
Tensor Operations

$$\tau_{ij}^{ab} = t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b,$$

$$\tilde{F}_e^m = f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f,$$

$$\tilde{F}_e^a = (1 - \delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} v_{ef}^{mn} t_{mn}^{af} + \sum_{fn} v_{ef}^{an} t_n^f,$$

$$\tilde{F}_i^m = (1 - \delta_{mi}) f_i^m + \sum_e \tilde{F}_e^m t_i^e + \frac{1}{2} \sum_{nef} v_{ef}^{mn} t_{in}^{ef} + \sum_{fn} v_{if}^{mn} t_n^f,$$



$$\tilde{W}_{ei}^{mn} = v_{ei}^{mn} + \sum_f v_{ef}^{mn} t_i^f,$$

$$\tilde{W}_{ij}^{mn} = v_{ij}^{mn} + P_j^i \sum_e v_{ie}^{mn} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{mn} \tau_{ij}^{ef},$$

$$\tilde{W}_{ie}^{am} = v_{ie}^{am} - \sum_n \tilde{W}_{ei}^{mn} t_n^a + \sum_f v_{ef}^{ma} t_i^f + \frac{1}{2} \sum_{nf} v_{ef}^{mn} t_{in}^{af},$$

$$\tilde{W}_{ij}^{am} = v_{ij}^{am} + P_j^i \sum_e v_{ie}^{am} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{am} \tau_{ij}^{ef},$$

$$z_i^a = f_i^a - \sum_m \tilde{F}_i^m t_m^a + \sum_e f_e^a t_i^e + \sum_{em} v_{ei}^{ma} t_m^e + \sum_{em} v_{im}^{ae} \tilde{F}_e^m + \frac{1}{2} \sum_{efm}$$

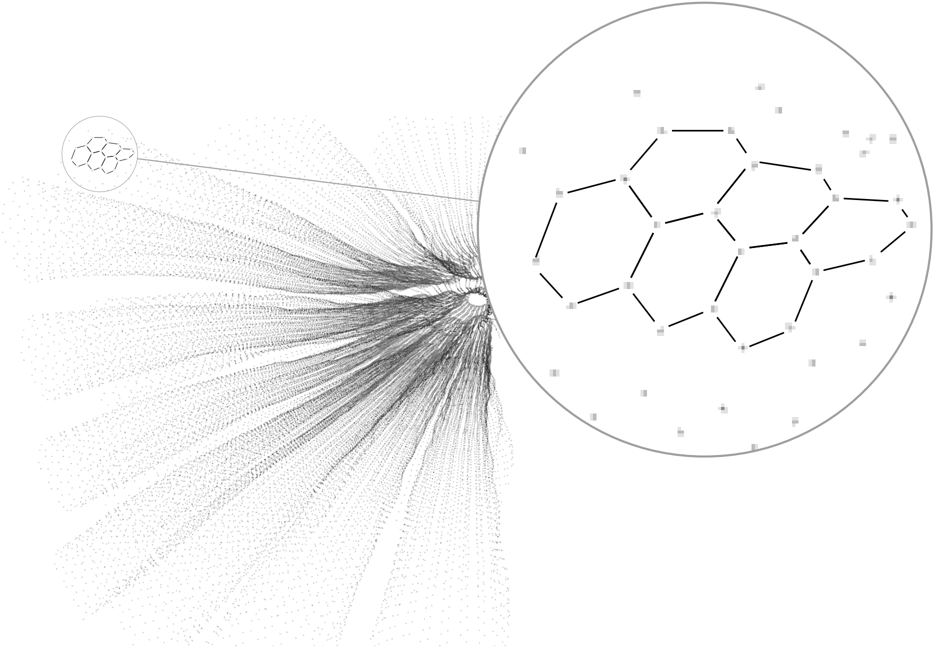
$$z_{ij}^{ab} = v_{ij}^{ab} + P_j^i \sum_e v_{ie}^{ab} t_j^e + P_b^a P_j^i \sum_{me} \tilde{W}_{ie}^{am} t_{mj}^{eb} - P_b^a \sum_m \tilde{W}_{ij}^{am} t_m^b + P_b^a$$

credits to D. Matthews, E. Solomonik, J. Stanton, and J. Gauss

P. Springer

- Tensor Transpositions Compiler TTC – github.com/HPAC/TTC
- High-Perf. Tensor Transp. Library HPTT – github.com/HPAC/HPTT
- Tensor Contraction Code Generator TCCG – github.com/HPAC/TCCG
- Tensor Contraction Library TCL – github.com/springer13/tcl
- TBLIS – by D. Matthews github.com/devinamatthews/tblis

Fast & portable code for complex molecular dynamics simulations



Molecular Dynamics Potentials

Name	**LOC-R**	**LOC-O**	**Structure**
LJ	640	+480	$\sum_i \sum_j f(i,j)$
Stillinger-Weber	600	+1250	$\sum_i \sum_j \sum_k f(i,j,k)$
EAM	840	+820	$\sum_i f(i, \sum_j g(i,j))$
Tersoff	800	+1450	$\sum_i \sum_j f(i,j, \sum_k g(i,j,k)$
MEAM	890	✗	$\sum_i f(i, \sum_j g(i,j))$
ADP	940	✗	too complex to show
BOP	5950	✗	too complex to show
(AI)REBO	4240	+4550	too complex to show
COMB3	3560	✗	too complex to show
ReaxFF	10880	✗	too complex to show

LOC-R: Lines of code of the regular code.

LOC-O: Extra LOC in the optimized/vectorized code.

Formalism: Pair vs Many-body

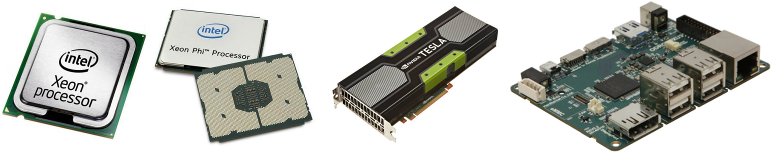
$$V = \sum_i \sum_j V(i, j)$$

$$F_i = -\nabla_{x_i} V$$

```
for i in atoms:
    for j in neighbors(i):
        V += V(i, j)
        F[i] -= dVdi(i, j)
        F[j] -= dVdj(i, j)
```

$$V = \sum_i \sum_j V(i, j, \sum_k f(i, j, k))$$

```
for i in atoms:
  for j in neighbors(i):
    tmp = 0
    for k in neighbors(i):
      tmp += f(i, j, k)
    V += V(i, j, tmp)
    F[i] -= dVdi(i, j, tmp)
    F[j] -= dVdj(i, j, tmp)
    tmp = dVdf(i, j, tmp)
    for k in neighbors(i):
      F[i] -= tmp * dfdi(i, j, k)
      F[j] -= tmp * dfdj(i, j, k)
      F[k] -= tmp * dfdk(i, j, k)
```



2013, 2014, 2015, 2016, 2017, 2018, 2019, . . .

One trend for Intel, ARM: Longer *vectors*

Need: Formulate program right, across differing HW

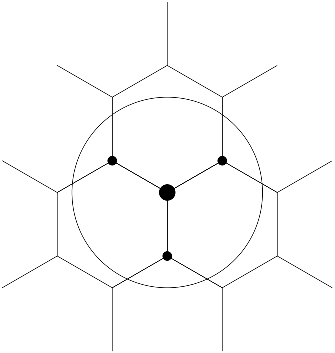
Everywhere...

x86	SSE	128 bit
	AVX(2)	256 bit
	AVX-512 (IMCI)	512 bit
ARM	NEON	128 bit
	SVE	up to 2048 bit
POWER	AltiVec/VMX/VSX	128 bit
	QPX	256 bit
SPARC	HPC-ACE	128 bit
	HPC-ACE2	256 bit

Manual Optimization: Tersoff

$\boxed{\text{Si}}\ \boxed{\text{C}}\ \boxed{\text{O}}\ \boxed{\text{Ga}}\ \boxed{\text{Ge}}\ \boxed{\text{B}}\ \boxed{\text{N}}$

$$V = \sum_i \sum_j V(i, j, \sum_k f(i, j, k))$$



Tersoff: Optimizations

Vector class library

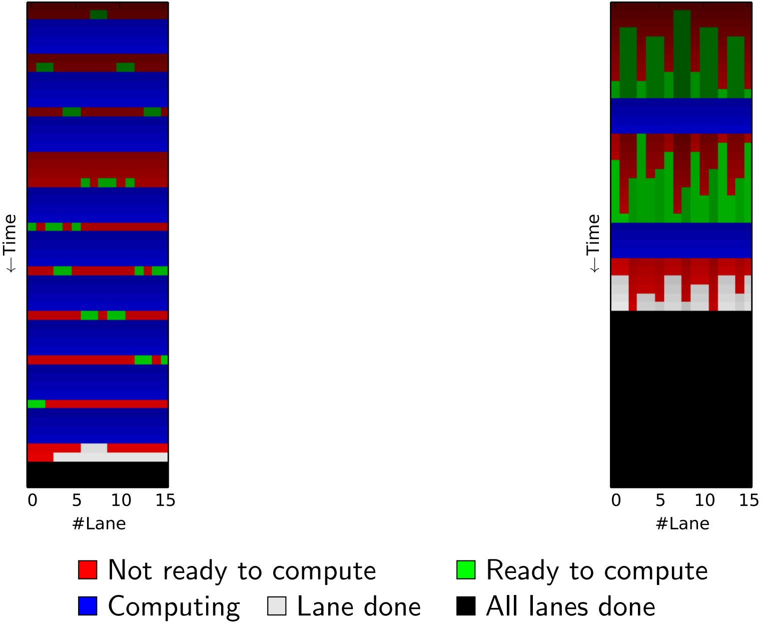
Reuse inner (k) calculation

Vectorization schemes: assign i, assign j, assign i and j

Fast forwarding

Neighbor list filtering

Tersoff: Fast forwarding



Tersoff: Neighbor list filtering

We iterate over the neighbor list multiple times.

Each time, we check whether the atom is within the cutoff

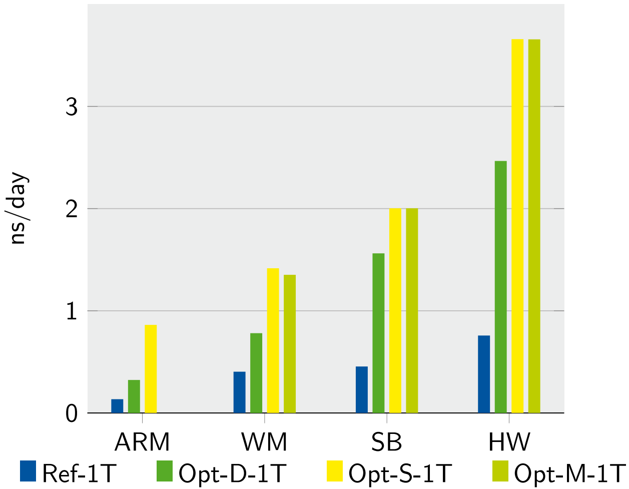
Redundant: Only do once, store the ones within cutoff

```
for (int i = ...) {
  if (foo(i)) continue;
  bar_1(i); }
bar_2();
for (int i = ...) {
  if (foo(i)) continue;
  bar_3(i); }
```

```
int n = 0, arr[MAX_N];
for (int i = ...) {
  if (foo(i)) continue;
  arr[n++] = i; }
for (int i = 0..n) {
  bar_1(arr[i]); }
bar_2();
for (int i = 0..n) {
  bar_3(arr[i]); }
```

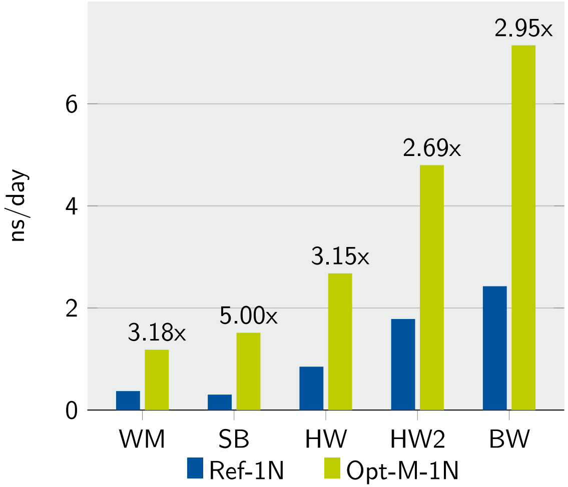
This technique has been adopted in LAMMPS

CPU: Single-Threaded Execution (32 000 atoms)



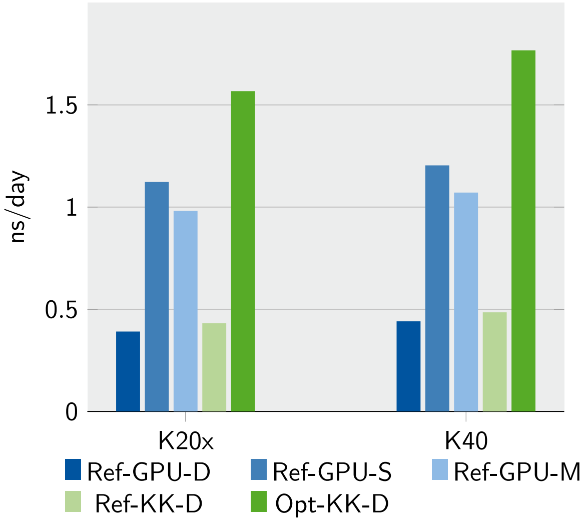
Name	Processor	Cores	Vector ISA
ARM	ARM Cortex-A15	2 × 4[1]	NEON
WM	Intel Xeon X5675	2 × 6	SSE4.2
SB	Intel Xeon E5-2450	2 × 8	AVX
HW	Intel Xeon E5-2680v3	2 × 12	AVX2

CPU: Single Node Execution (512 000 atoms)



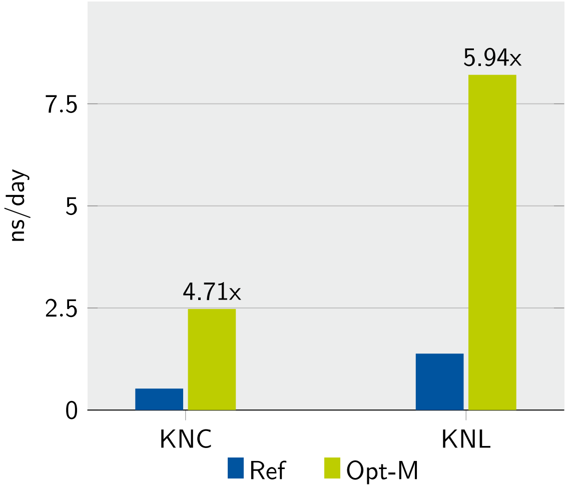
Name	Processor	Cores	Vector ISA
WM	Intel Xeon X5675	2 × 6	SSE4.2
SB	Intel Xeon E5-2450	2 × 8	AVX
HW	Intel Xeon E5-2680v3	2 × 12	AVX2
HW2	Intel Xeon E5-2697v3	2 × 14	AVX2
BW	Intel Xeon E5-2697v4	2 × 18	AVX2

GPU (256 000 atoms)



Name	CPU	Cores	ISA	Accelerator
K20X	Intel Xeon E5-2650	2 × 8	AVX	Nvidia Tesla K20x
K40	Intel Xeon E5-2650	2 × 8	AVX	Nvidia Tesla K40

Native Execution on Xeon Phi Systems (512 000 atoms)



Name	CPU	Cores	ISA	Accelerator	Cores	ISA
IV+2KNC	Intel Xeon E5-2650v2	2 × 8	AVX	Intel Xeon Phi 5110P	2 × 60	IMCI
KNL	–	–	–	Intel Xeon Phi 7250	68	AVX-512

Manual Optimization: AIREBO

$$\boxed{\text{C}}\ \boxed{\text{H}}$$

$$V = \sum_i \sum_j^i V(i, j, \sum_k^i f(i,k), \sum_l^j f(j,l), \sum_k^i \sum_l^j g(i,j,k,l))$$

$$+ \sum_i \sum_j^i \sum_k^i \sum_l^j V'(i,j,k,l)$$

$$+ \sum_i \sum_j^{i'} V''(i, j, \max g(i,j,k,l) \forall k, l, \sum_k^i f(i,k), \sum_l^j f(j,l), \sum_k^i \sum_l^j g(i,j,k,l))$$

Two neighborhoods: Small and large ($\sum^i$ vs $\sum^{i'}$)

AIREBO: Optimizations

Intermediate neighbor list

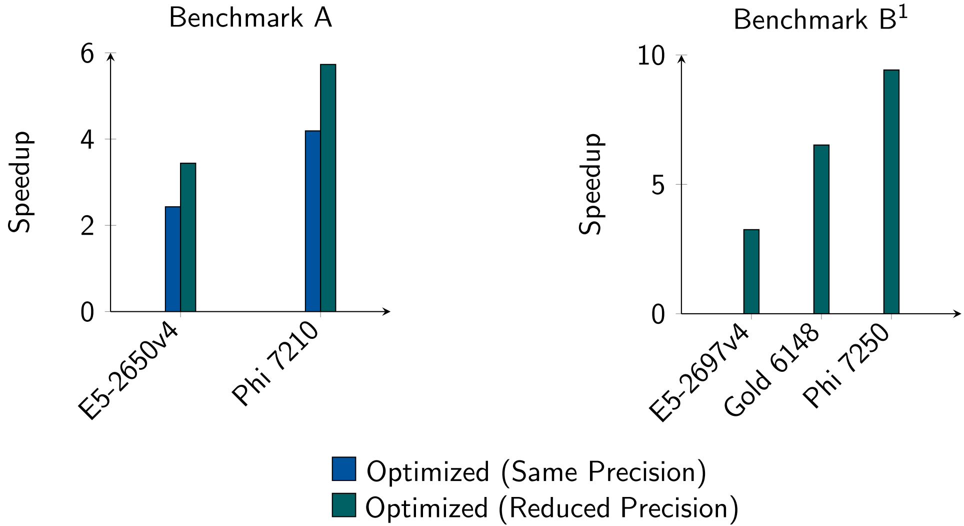
Compression

Batching by atom species

Reuse of intermediate values

Global search and lookup (hash-map) instead of local search

AIREBO: Results



[1] M. Brown (Intel): Speedups from http://lammps.sandia.gov/doc/accelerate_intel.html, Oct. 2017.

What's wrong with manual optimization?

N optimization techniques (we saw about 5 of them)

M hardware platforms (we saw more than 5 of them)

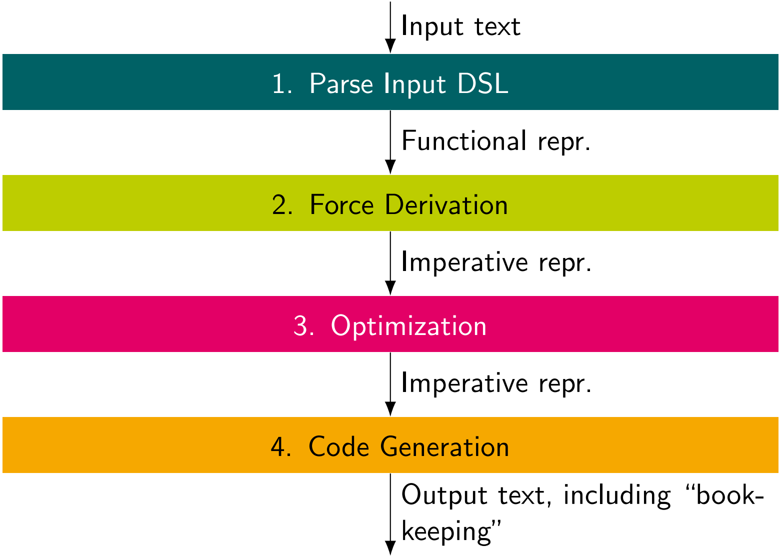
K potentials (we saw 10, but there are many more)

Without platform abstraction: $N \times M \times K$ implementations

With platform abstraction: $N \times K$ implementations

But: Technique may depend on HW platform...

Our approach: PotC, A domain-specific language and a compiler



Input text

1. Parse Input DSL

Functional repr.

2. Force Derivation

Imperative repr.

3. Optimization

Imperative repr.

4. Code Generation

Output text, including “book-keeping”

More advantages

Not *best* performane, but *good* performance

Portability

Ease of implementation

Coverage

Prototyping

Testing

Mechanical exploration (Profiling/Precision)

Other symbolic manipulations

DSL: Overview

```
parameter A(i: atom_type; j: atom_type) = file(1);
function V(i: atom; j: atom) = ...;
energy sum(i: all_atoms)
         sum(j: neighbors(A(i, j), i)) V(i, j);
```

DSL: Sums

```
sum(<index>:  all_atoms) ...

sum(<index>:  neighbors(<cutoff>, <center_atom>[,
exclusions]) ...

sum(<index>:  neighbors_half(...)  ...

sum(i: all_atoms) sum(j: neighbors(r_C(i, j), i)) V(i, j)
```

DSL: Parameters

```
parameter <name>({<name>:  atom_type;}*) = file(<n>);
```

```
parameter <name>({<name_i>:  real;}*)=
spline(<name>, <degree>, {<dimension_i>}*[,
<derivative_spec>]);
```

DSL: Functions

```
function <name>({<param>: <type>;}*) = <expr>;
```

```
function abs(x: real) = sqrt(x*x);
```

DSL: Other features

```
piecewise({<cond>: <val>;}*)

function abs(x: real) = piecewise(x < 0: -x; x >= 0: x);

let(<name>: <val>)

function abs(x: real) = let(a: x*x) sqrt(a);
```

DSL: Putting it all together

```
energy 1 / 2 * sum(i : all_atoms)
  sum(j : neighbors(i, R(i, j, j) + D(i, j, j))) V(i, j);
function V(i : atom; j : atom) =  f_C(i, j, r(i, j)) *
 (f_R(i, j, r(i, j)) + b(i, j) * f_A(i, j, r(i, j)));
function f_C(i : atom_type; j : atom_type; k : atom_type; r : distance) =
 implicit(i : i; j : j; k : k) piecewise(r <= R - D : 1;
 R - D < r < R + D: 1 / 2 - 1 / 2 * sin(pi / 2 * (r - R) / D); r >= R + D : 0);
# ...
function b(i : atom; j : atom) = (1 + beta(i, j) ^ n(i, j) *
  zeta(i, j) ^ n(i, j)) ^ (-1 / (2 * n(i, j)));
function zeta(i : atom; j : atom) =
 sum(k : neighbors(i, R(i, j, k) + D(i, j, k), j))
 implicit(i : i; j : j; k : k) f_C(r(i, k)) * g(theta(j, i, k)) *
   exp(lambda_3 ^ m * (r(i, j) - r(i, k)) ^ m);
parameter A(i : atom_type; j : atom_type) = file(1);
# ...
parameter gamma(i : atom_type; j : atom_type; k : atom_type) = file(5);
# Total: ~ 30 LOC, while LAMMPS impl > 500 LOC
```

Optimization: Overview

The initial code is far from optimal since it recomputes a lot of values.

No tape due to memory constraints.

Inlining

Loop/Conditional Fusion

Dead Code Elimination

Common Subexpression Elimination

Arithmetic Improvements (e.g Constant Folding)

Loop Invariant Code Motion

Vectorization

Check for 0

Caching

Batching

Optimization: CSE and LICM

Two techniques commonly used by any compiler.

Common Subexpression Elimination

Loop Invariant Code Motion

Compilers often are not sophisticated enough to move loop invariant control flow out

```
double a = foo(x);             double a = foo(x);
double b = foo(x) * bar(y);    double b = a * bar(y);

for (int i = ...) {            double a = foo(x);
  double a = foo(x);           for (int i = ...) {
  ...                            ...
}                              }
```

Optimization: Neighbor list filtering

We used this for both Tersoff and AIREBO

One of the crucial techniques that compilers are unlikely to ever perform

Applied as soon as a neighbor list is traversed more than once

```
for (int i = ...) {
  if (foo(i)) continue;
  bar_1(i); }
bar_2();
for (int i = ...) {
  if (foo(i)) continue;
  bar_3(i); }
```

```
int n = 0, arr[MAX_N];
for (int i = ...) {
  if (foo(i)) continue;
  arr[n++] = i; }
for (int i = 0..n) {
  bar_1(arr[i]); }
bar_2();
for (int i = 0..n) {
  bar_3(arr[i]); }
```

Optimization: Vectorization

Force the compiler: We know it will work

Apply the fast-forwarding logic we used for both AIREBO and Tersoff

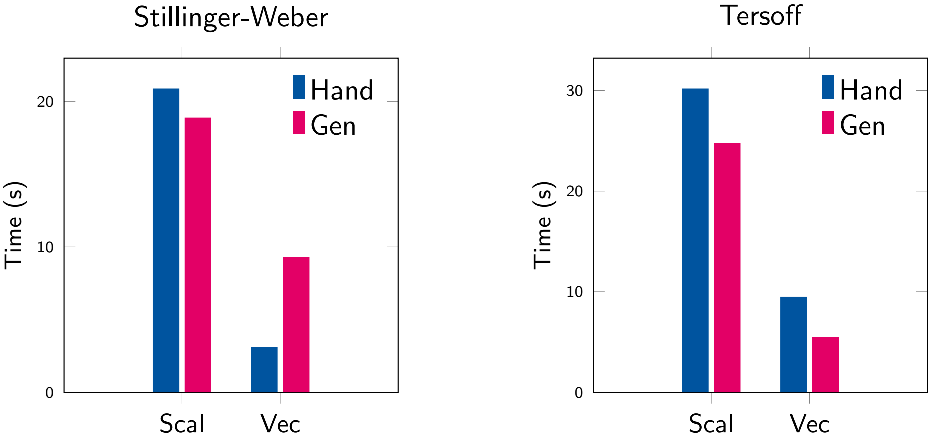
Assumption: The check is much cheaper than the calculation

Can perform idioms compilers do not yet support without intrinsics

```
for_VEC (int i=...) {
  for (int j=...) {
    if (foo(i, j)) continue;
    bar(i, j);
  }
}
```

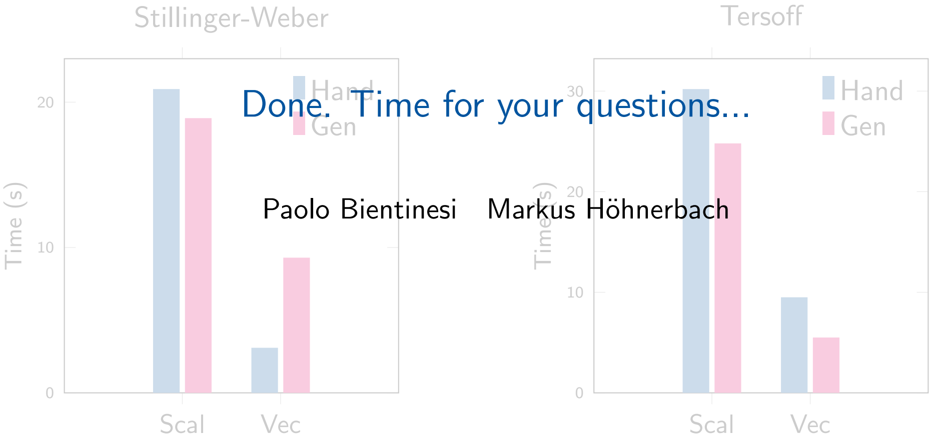
```
for_VEC (int i=...) {
  for (int j=...) {
    if (ANY(foo(i, j))) {
      if (foo(i, j)) {
        j += 1; }
      continue; }
    bar(i, j);
  }
```

Results



KNL, double precision, single core.

Results



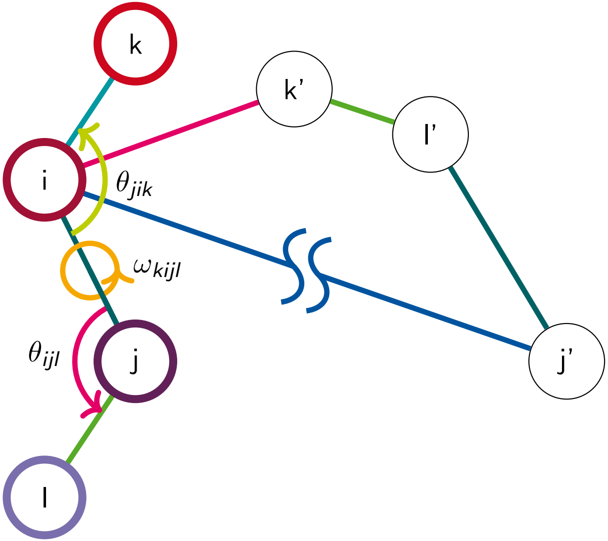
Done. Time for your questions...

Paolo Bientinesi   Markus Höhnerbach

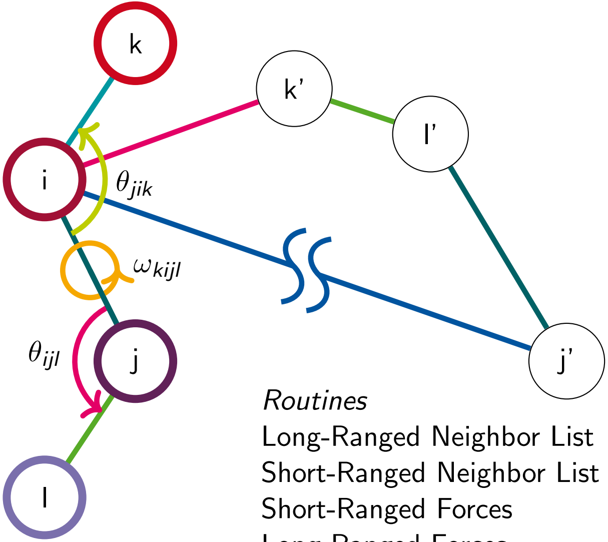
KNL, double precision, single core.

Backup

AIREBO



AIREBO



Routines

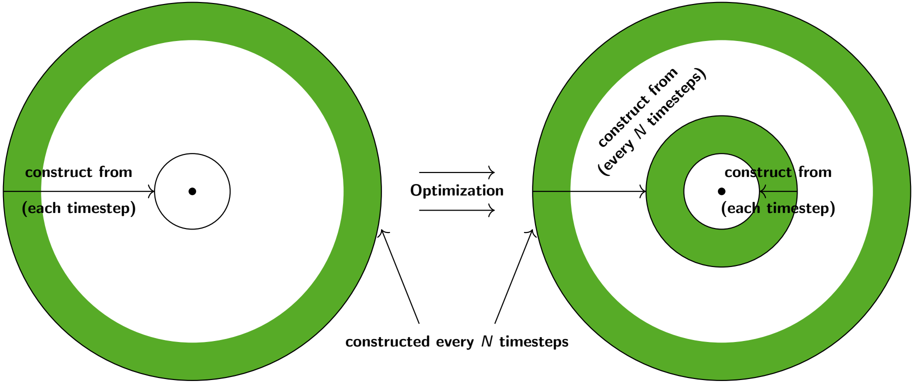
Long-Ranged Neighbor List

Short-Ranged Neighbor List

Short-Ranged Forces

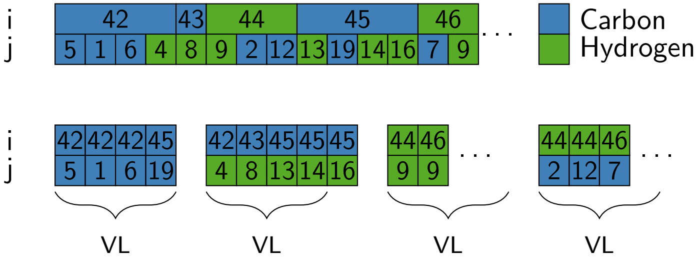
Long-Ranged Forces

Short-Ranged Neighbor List



Consider fewer atoms for short neighbor list each timestep.

Short-Ranged Contributions: Sorting



i	42				43	44			45				46		. . .
j	5	1	6	4	8	9	2	12	13	19	14	16	7	9	

🟦 Carbon

🟩 Hydrogen

i	42	42	42	45
j	5	1	6	19

VL

i	42	43	45	45	45
j	4	8	13	14	16

VL

i	44	46
j	9	9

. . .

VL

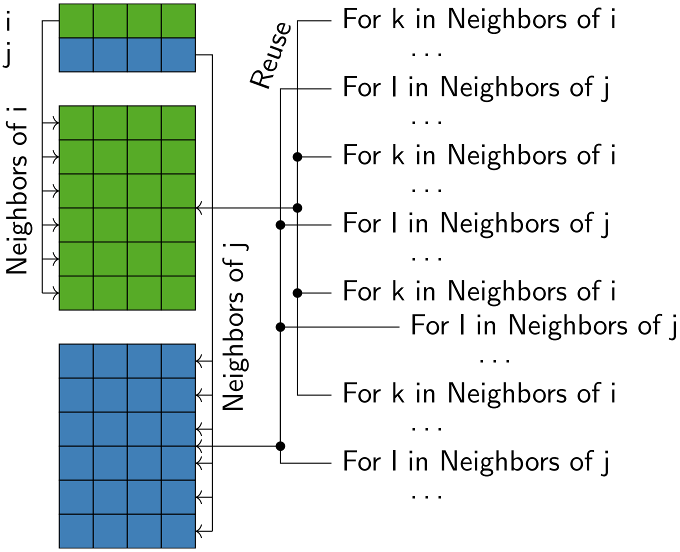
i	44	44	46
j	2	12	7

. . .

VL

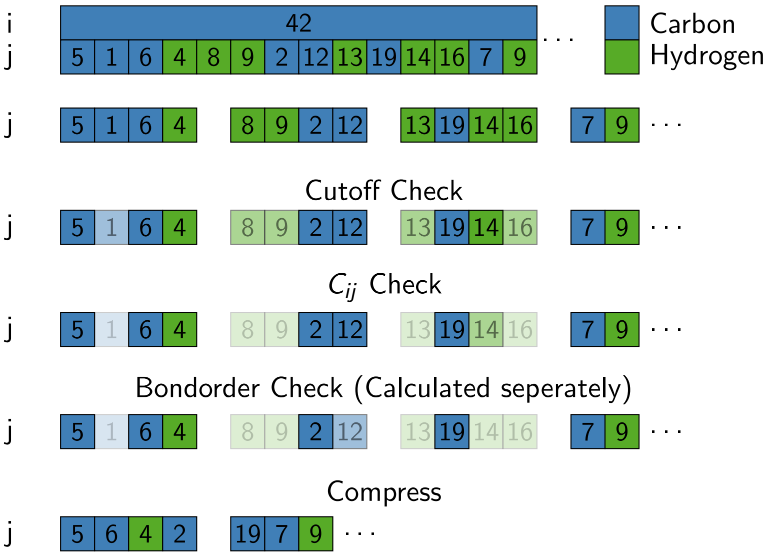
Vectorize over interactions and bin by species.

Short-Ranged Contributions: Reuse



```
For k in Neighbors of i
    . . .
For l in Neighbors of j
    . . .
For k in Neighbors of i
    . . .
For l in Neighbors of j
    . . .
For k in Neighbors of i
    For l in Neighbors of j
        . . .
For k in Neighbors of i
    . . .
For l in Neighbors of j
    . . .
```

Long-Ranged Contributions: Compression



i 42 · · ·

j 5 1 6 4 8 9 2 12 13 19 14 16 7 9

🟦 Carbon

🟩 Hydrogen

j 5 1 6 4 | 8 9 2 12 | 13 19 14 16 | 7 9 · · ·

Cutoff Check

j 5 1 6 4 | 8 9 2 12 | 13 19 14 16 | 7 9 · · ·

C_{ij} Check

j 5 1 6 4 | 8 9 2 12 | 13 19 14 16 | 7 9 · · ·

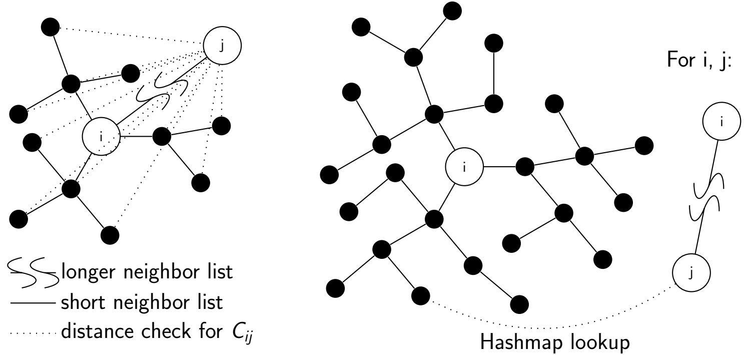
Bondorder Check (Calculated seperately)

j 5 1 6 4 | 8 9 2 12 | 13 19 14 16 | 7 9 · · ·

Compress

j 5 6 4 2 | 19 7 9 · · ·

Long-Ranged Contributions: Hashmap



For i, j: Search neighbors.

For i: Insert candidates into hashmap.

For i, j:

longer neighbor list

short neighbor list

distance check for C_{ij}

Hashmap lookup

Precompute hashmap to replace search with lookups.