LA-UR-17-22676
Approved for public release; distribution is unlimited.

Title: A Case Study on Software Modernization using CoMD - A Molecular Dynamics Proxy Application

Author(s): Adedoyin, Adetokunbo

Intended for: colfax research webinar, 2017-04-04 (los alamos, New Mexico, United States)

Issued: 2017-04-03

Disclaimer:
Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

A Case Study on Software Modernization using CoMD - A Molecular Dynamics Proxy Application

by
Dr. Adedoyin
toks@lanl.gov

https://colfaxresearch.com/mc2-series/

About Me:

- Current:
  - Specializing in Software Modernization
  - Future Application and Architectures (FAA)
  - Application Performance Team (APT)
- Past:
  - Post doctoral Fellow (ND)
    - Developed crystal plasticity framework for c-BN Synthesis via shockwave processing
  - Computational Solid Mechanics(CSM)
    - Modeling phase transformation in solids
  - Computational Fluid Dynamics (CFD)
    - Turbulence modeling (real/spectral space)

Gratitude:

- DoE
- LANL Institutional Computing
- Additional Contributions:
  - Cray(Hackaton/Boot-Camp)
  - Intel(Hackaton/Boot-Camp)
- LANL Folks:
  - CoMD Development Team
  - Colleagues

This Talk I:

Performance Lingo

Pre Software Modernization

- In thinking about software modernization
- Tools – "The Necessary Evil"
- A Modernization Mindset - The Cutting Rod Approach
- Roofline Analysis – In a Nutshell
- Roofline Analysis – The Outcome

This Talk II:

Software Modernization Case Study 2 – CoMD

CoMD:

- Algorithm Description
- Driver Blue Print
- Time Marching
- Force Kernels
- Modernization Goals
- L-J Force Kernel Optimization
- EAM Force Kernel Optimization

Post Software Modernization:

- Comparing: HSW vs. KNL, How?

Performance Lingo:

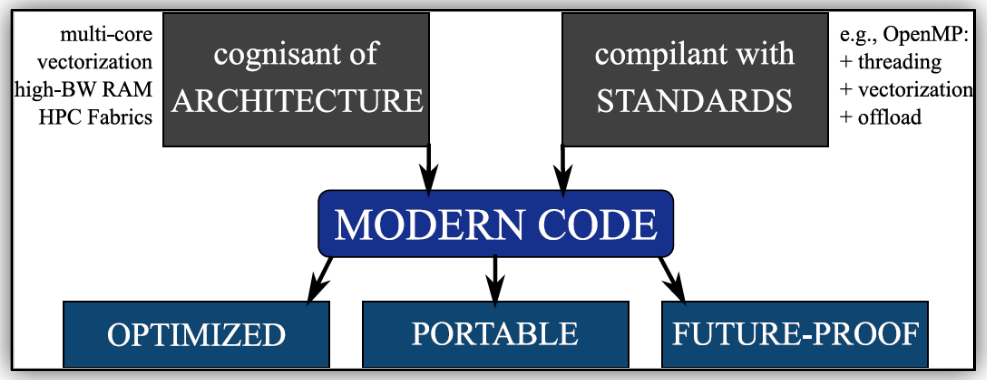
- TDP – Thermal Design Power
- HSW – Haswell (Intel processor)
- KNL – Knights Landing (Intel Xeon Phi 2nd Generation)
- SNB – Sandy Bridge (Intel processor)
- IVY – IVY Bridge (Intel processor)
- BWD – Broadwell Bridge (Intel processor)
- SIMD – Single Instruction Multiple Data
- MPI – Message Passing Interface
- OpenMP – Open Multi-Processing (Thread Parallel Paradigm)
- Vectorization – Vector representation of similar
  scalar operations (Data Parallelization)
- Directive/Primitive -

This Section discusses …

PRE
SOFTWARE MODERNIZATION

In Thinking about Software Modernization:

- Why? Hardware is a rate quantity:
  - H’=f(Technology)
- Hiding Communication Primitives
  - c++: communication classes
  - c: communication source files
  - Fortran: communication modules
- Hiding Threading Primitives
  - Pragmas&/Directives
  - Challenging!
- Hiding Vectorization Primitives
  - Pragmas&/Directives
  - Challenging!
- Hiding Alignment/Allocation Primitives
  - aligned malloc:
    - __aligned__malloc(,)
    - posix_align(,) (still in dev.)
  - align compiler hint: __mm__aligned(,)
  - align size: #ifdef <MachineType> ALIGNMENT_SIZE 64



Modern code graphic – colfax research <https://colfaxresearch.com/>

Tools – The Necessary Evil:

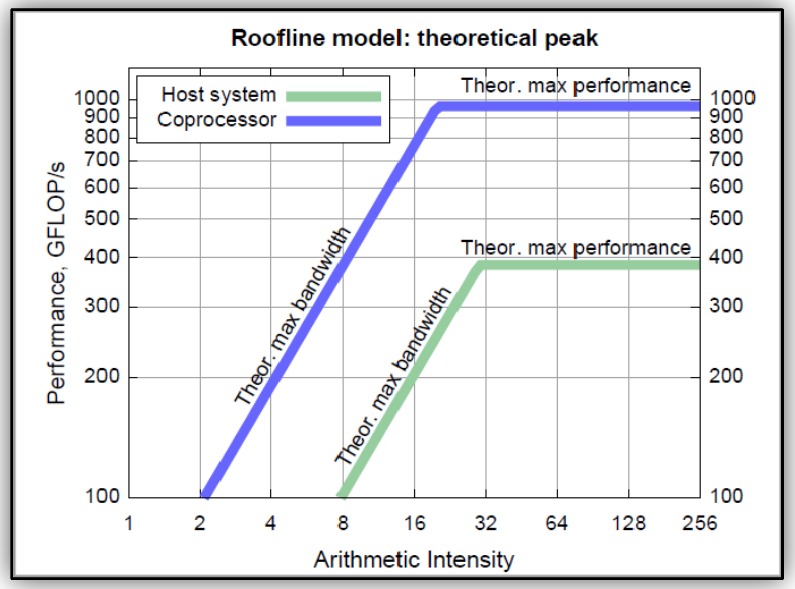
- Understand
- Intel® Advisor
  - Analysis kernels
  - Roofline
- Intel® Vtune™ Amplifier
  - Analysis kernels
- Allinea(ddt/map)

A Modernization Mindset - The Cutting Rod Approach:

- Goal:
  - Determine optimal substructure.
  - Combined optimal substructures will probably be global optimum.

- Software modernization is:
  - Iterative (pick your poison)
  - Exhaustive(work smart):
    - Use Proxy/Mini App (~20K lines)
    - Try Multiples approaches
- Note:
  - Optimal code for KNL does not translate to optimal code either HSW/SNB/BWD

Roofline Analysis – In a Nutshell:



roofline model graphic - colfax research <https://colfaxresearch.com/>

Family of functions

Axis:

- Abscissa:
  - Flops/DRAM-Byte
  - Relative CPU to Memory utilization.
- Ordinate:
  - Flops/s

Roofline Analysis – The Outcome:

Roofline results:

- Arithmetic Intensity (AI)
  - What is thread affinity for max performance?
  - What is max number of threads for max performance?
  - Do you need to incorporate:
    - Vectorization (Relative Quality/Quantity)?
    - Prefetching (primitives/compiler flags)…?
    - Data Contiguity…?
- Does your code need:
  - Re-timing?
  - Dynamic thread affinity?
- Does your code need heavy rewrite to change the AI?

This Section discusses …

CASE STUDY: CoMD SOFTWARE MODERNIZATION

CoMD – Algorithm Description:

- **CoMD** Open-Source Molecular Dynamics Proxy Application:
  - Molecular dynamics is a low level “higher resolution” material modeling approach.
  - https://github.com/exmatex/CoMD
- Types of Force Kernels:
  - Lenard Jones (L-J)
  - Embedded Atom Model (EAM)
- Code Branches:
  - Serial (focus for data-parallel)
  - OpenMP (focus for data & thread parallel)
    - Loop level implementation
  - MPI / MPI+OpenMP
- Problem Type:
  - N-Body
- Decomposition Type:
  - Cartesian
- Complexity:
  - O(~n^2)

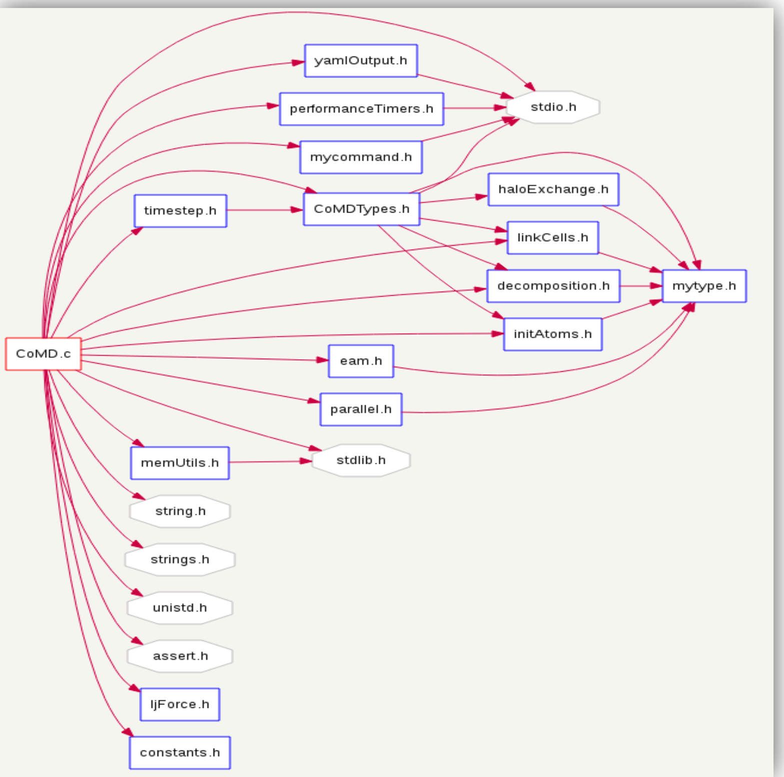
CoMD – Example Application:

- An approach to computationally fabricating c-BN via shock-wave processing will require modeling a material domain at impact.
- Upon impact, the material domain will categorically contain multiple activity regions:
  - Shock Zone – Higher activity region
  - Transition Zone – Mid activity region
  - Inert Zone – Low activity region
    - (chemically/mechanically relatively inactive)
- Molecular dynamics is a theoretically feasible approach to capturing the underlying physics (the essence) in the material’s shock zone.
  - A relatively higher level mid resolution may be sufficient!
    - i.e. Crystal plasticity

CoMD – Driver Blue-Print

Understand View:

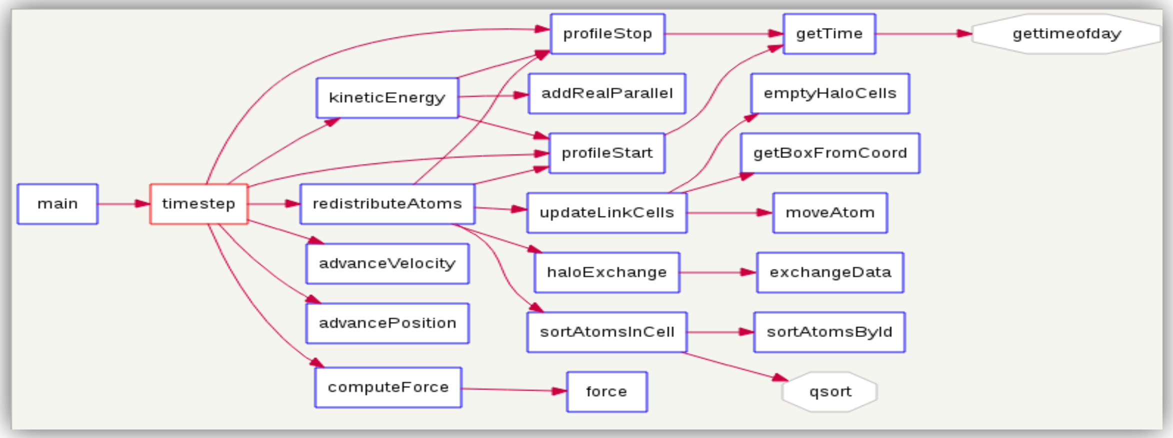
- Infinite call depth:
  - Connectivity of the code
- Blue-square imply:
  - Subroutines/function/headers
- Gray-hexagon imply:
  - c-libs



CoMD - Time Marching:

Understand View:

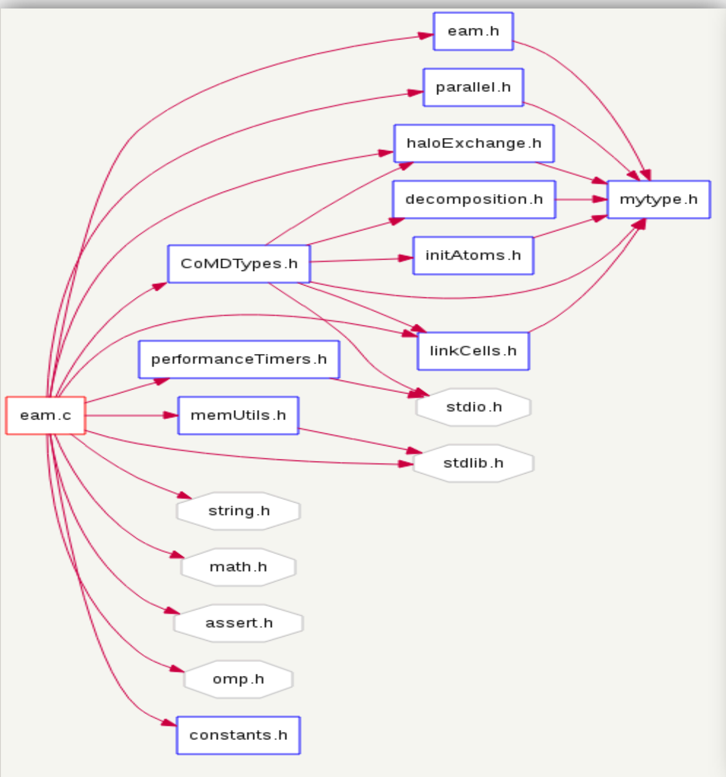
- Infinite call depth
- Blue Square imply:
    - subroutines/function
- Gray-hexagon imply:
    - c-libs

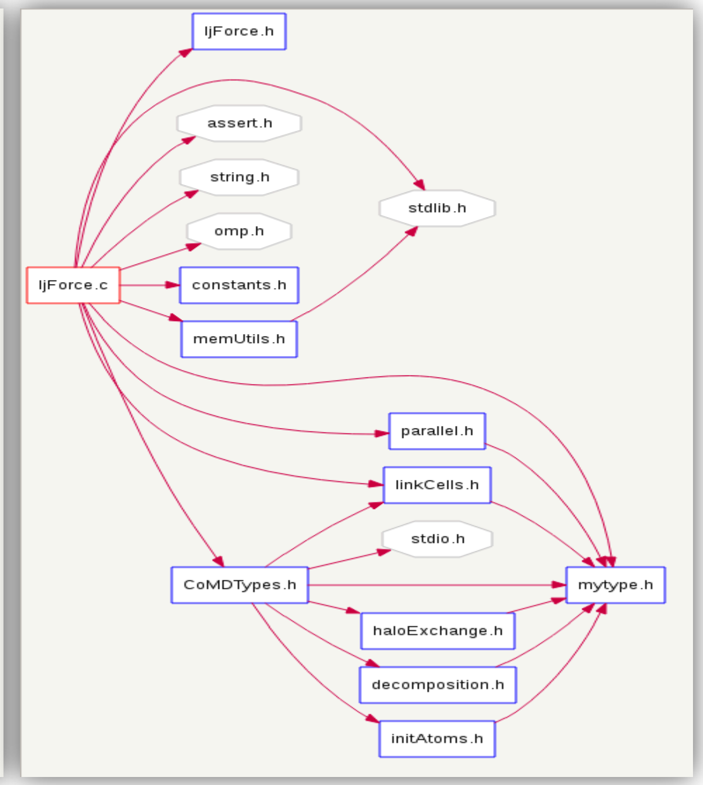


CoMD - Force Kernels:

Force kernels:

- **EAM:**
  - 3-passes
- **LJ:**
  - 1-pass





CoMD – Modernization

- Modernization Mindset:
  - “Cutting Rod Approach”
  - Optimal substructures
- Machine Choice (KNL/64-68cores/4-HT):
  - Based on Baseline runs
    - NUMA decomposition(SNC4/SNC2/QUAD)
    - Memory hierarchy(Flat/Cache)
- Modernization Exercise Goals:
  - Improve Vectorization
  - Improve Threading

App Runtime Comparison - KNL vs. HSW:

How?:

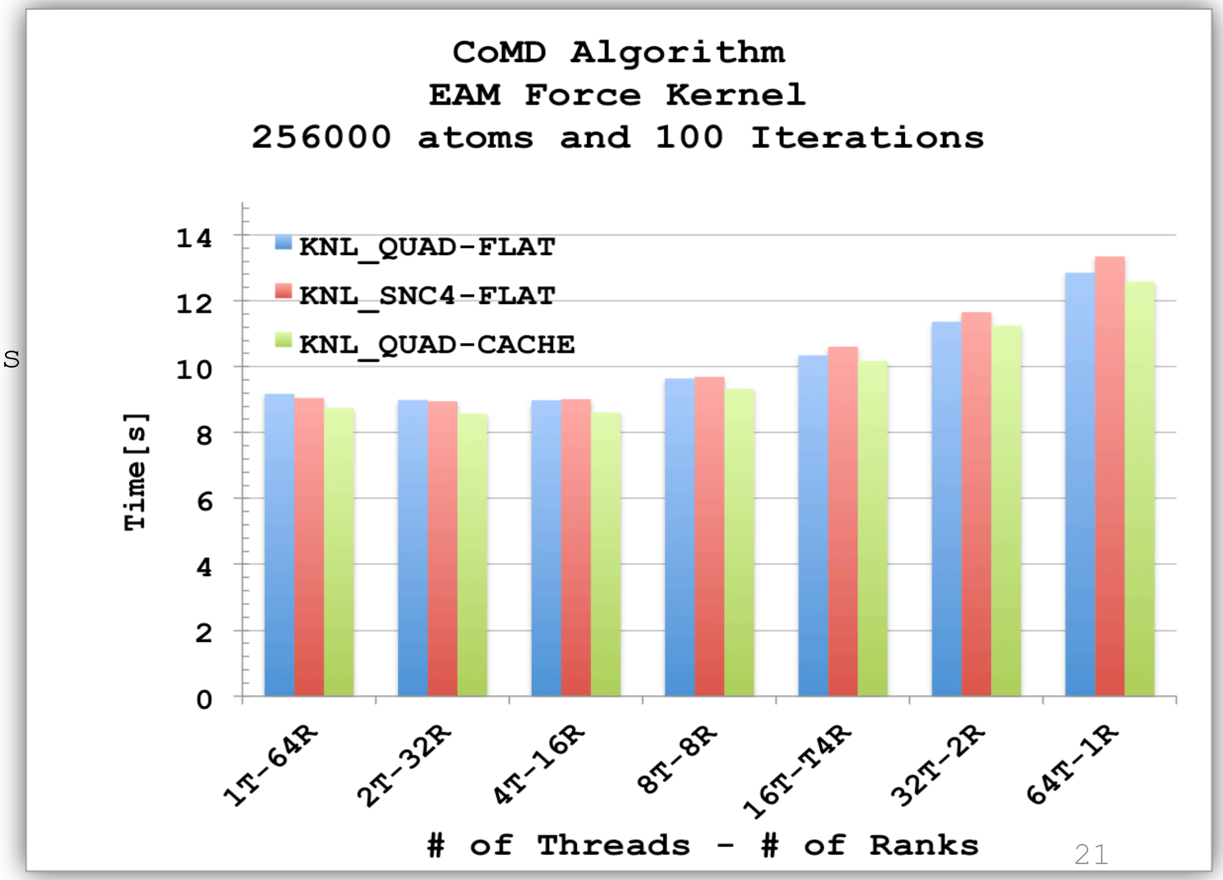
- TDP is a “useful” metric
- It is “Not” an exact science!
- Given:
  - Single-Node KNL
  - ~200W CPU TDP
- Choose HSW with comparable TDP:
  - Single-Node, Dual-Socket HSW
  - 2x(Intel® Xeon® processor E5-2697v3)
  - ~2x(145W CPU TDP)
- Intel Processor Specs:
  - ark.intel.com

CoMD – Baseline Runs I

CoMD Simulation Details:

KNL:

- Intel® Xeon Phi™ Processor 7210
- CoMD on Node performance
- Multiple MPI Ranks
- Complementary OpenMP Threads
- Best Results:
  - QUAD-Cache
  - 2-Threads
  - 32-Ranks
- Comparable Results:
  - QUAD-Cache
  - 1TH-64R
  - 4TH-16R

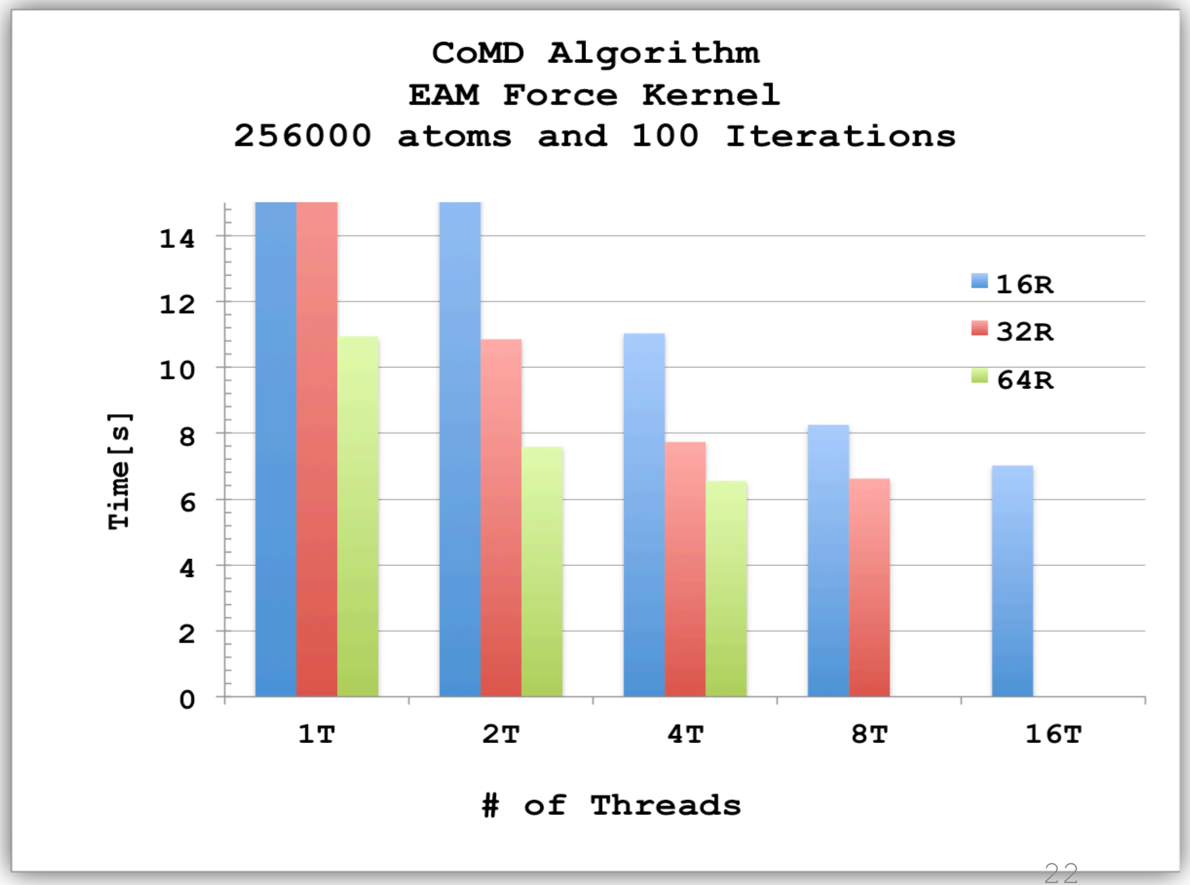


CoMD – Baseline Runs II

<u>CoMD Simulation Details:</u>

KNL:

- Intel® Xeon Phi™ Processor 7210
- CoMD on Node performance
- MPI Ranks (16/32/64)
- Complementary OpenMP Threads
- Best Results:
  - 64-Ranks
  - 32-Ranks
- Comparable Results:
  - QUAD-Cache
  - 1TH-64R
  - 4TH-16R

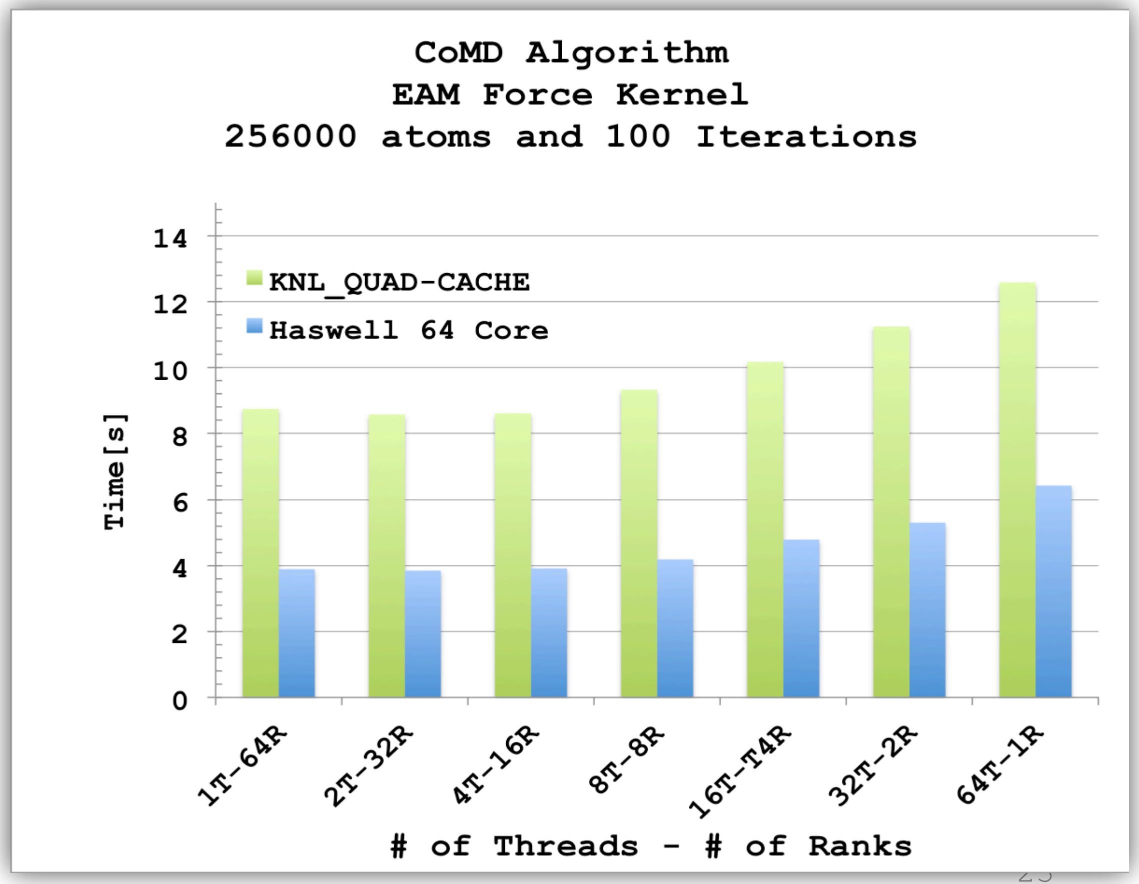


CoMD – Baseline Runs III

CoMD Simulation Details:

KNL:
- Intel® Xeon Phi™ Processor 7210

HSW - Dual Socket:
- Intel® Xeon® Processor E5-2697v3
- CoMD on Node performance
- MPI Ranks
- Complementary OpenMP Threads
- Best Results:
  - 32-Ranks/2TH
- Note:
  - Oversubscribing
    - HSW > 2TH
    - KNL > 4TH



CoMD – Hotspots

Intel® Vtune™ Amplifier:
- **hotspots analysis**

CoMD Performance Profiling:

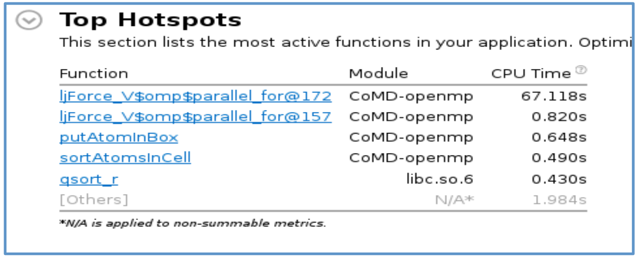
Intel® Vtune™ Amplifier:
- Hotspots kernel

- Summary:

CoMD with LJ Force:
1. LJ-Force(,,)
2. putAtomInBox(,,)

CoMD with EAM Force:
1. EAM-Force(,,)
2. sortAtomInCell(,,)

Summary:	L-J Force		EAM Force	
% Runtime Per Region	Serial:	~ 2%	Serial:	~ 1%
	Parallel:	~ 98%	Parallel:	~ 99%
% Per Subroutine Runtime	Lenard-Jones:	~ 93%	EAM:	~ 94%
	Others:	~ 7%	Others:	~ 6%

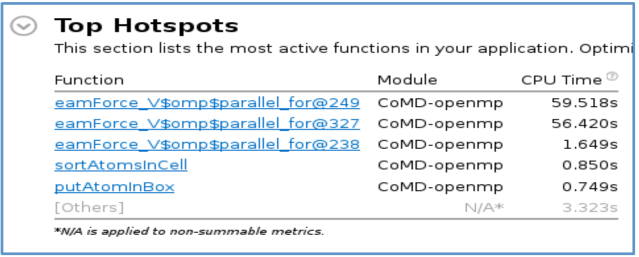


Top Hotspots

This section lists the most active functions in your application. Optimi

Function	Module	CPU Time
ljForce_V$omp$parallel_for@172	CoMD-openmp	67.118s
ljForce_V$omp$parallel_for@157	CoMD-openmp	0.820s
putAtomInBox	CoMD-openmp	0.648s
sortAtomsInCell	CoMD-openmp	0.490s
qsort_r	libc.so.6	0.430s
[Others]	N/A*	1.984s

*N/A is applied to non-summable metrics.



Top Hotspots

This section lists the most active functions in your application. Optimi

Function	Module	CPU Time
eamForce_V$omp$parallel_for@249	CoMD-openmp	59.518s
eamForce_V$omp$parallel_for@327	CoMD-openmp	56.420s
eamForce_V$omp$parallel_for@238	CoMD-openmp	1.649s
sortAtomsInCell	CoMD-openmp	0.850s
putAtomInBox	CoMD-openmp	0.749s
[Others]	N/A*	3.323s

*N/A is applied to non-summable metrics.

L-J Force – Simulation Output:

CoMD Problem Definition:

Force Kernel:

- Lennard – Jones

Number of Atoms:

- 32000

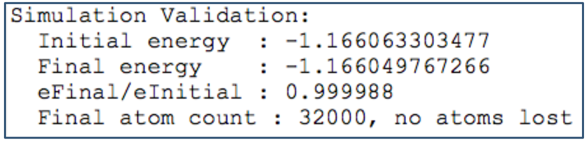
Number of TimeSteps:

- 100

Parallelization:

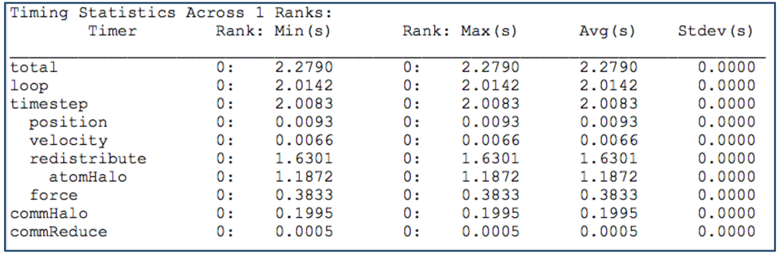
- MPI
- OpenMP

V&V Information:

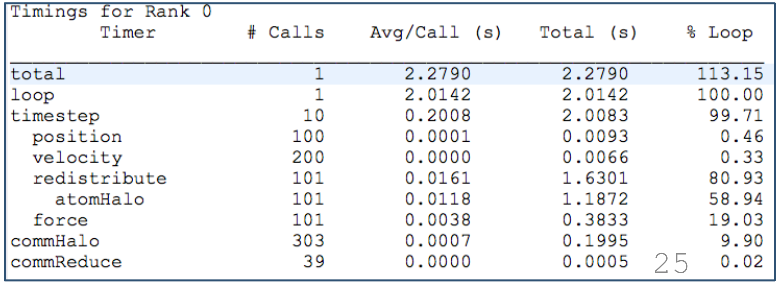


```
Simulation Validation:
  Initial energy  : -1.166063303477
  Final energy    : -1.166049767266
  eFinal/eInitial : 0.999988
  Final atom count : 32000, no atoms lost
```

Output:

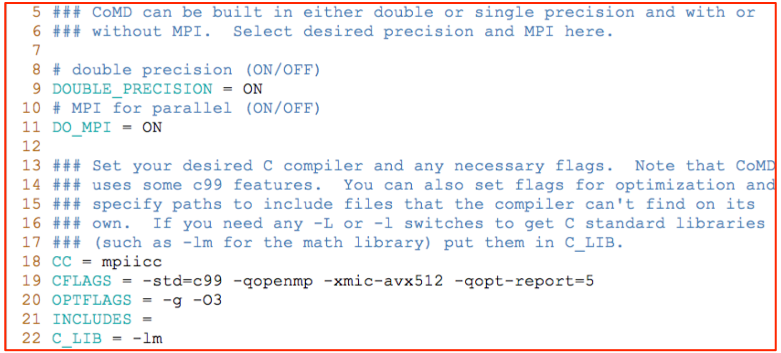


```
Timing Statistics Across 1 Ranks:
        Timer           Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total                    0:    2.2790       0:    2.2790      2.2790    0.0000
loop                     0:    2.0142       0:    2.0142      2.0142    0.0000
timestep                 0:    2.0083       0:    2.0083      2.0083    0.0000
  position               0:    0.0093       0:    0.0093      0.0093    0.0000
  velocity               0:    0.0066       0:    0.0066      0.0066    0.0000
  redistribute           0:    1.6301       0:    1.6301      1.6301    0.0000
    atomHalo             0:    1.1872       0:    1.1872      1.1872    0.0000
  force                  0:    0.3833       0:    0.3833      0.3833    0.0000
commHalo                 0:    0.1995       0:    0.1995      0.1995    0.0000
commReduce               0:    0.0005       0:    0.0005      0.0005    0.0000
```



```
Timings for Rank 0
        Timer        # Calls    Avg/Call (s)   Total (s)    % Loop
___________________________________________________________________
total                      1       2.2790        2.2790     113.15
loop                       1       2.0142        2.0142     100.00
timestep                  10       0.2008        2.0083      99.71
  position               100       0.0001        0.0093       0.46
  velocity               200       0.0000        0.0066       0.33
  redistribute           101       0.0161        1.6301      80.93
    atomHalo             101       0.0118        1.1872      58.94
  force                  101       0.0038        0.3833      19.03
commHalo                 303       0.0007        0.1995       9.90
commReduce                39       0.0000        0.0005       0.02
```

Build Information:



```
### CoMD can be built in either double or single precision and with or
### without MPI.  Select desired precision and MPI here.

# double precision (ON/OFF)
DOUBLE_PRECISION = ON
# MPI for parallel (ON/OFF)
DO_MPI = ON

### Set your desired C compiler and any necessary flags.  Note that CoMD
### uses some c99 features.  You can also set flags for optimization and
### specify paths to include files that the compiler can't find on its
### own.  If you need any -L or -l switches to get C standard libraries
### (such as -lm for the math library) put them in C_LIB.
CC = mpiicc
CFLAGS = -std=c99 -qopenmp -xmic-avx512 -qopt-report=5
OPTFLAGS = -g -O3
INCLUDES =
C_LIB = -lm
```

Source Code: CoMD

LJ-Force Kernel

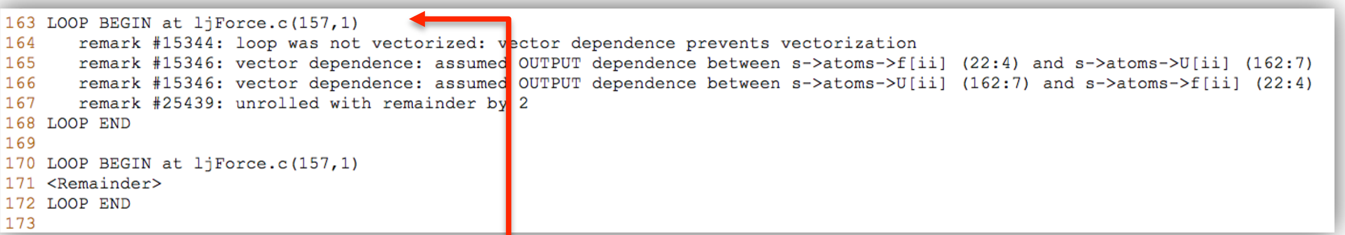
Initialization Step

L-J Force – Baseline:

Compiler:

- Intel 17/up1

Optimization Reports:



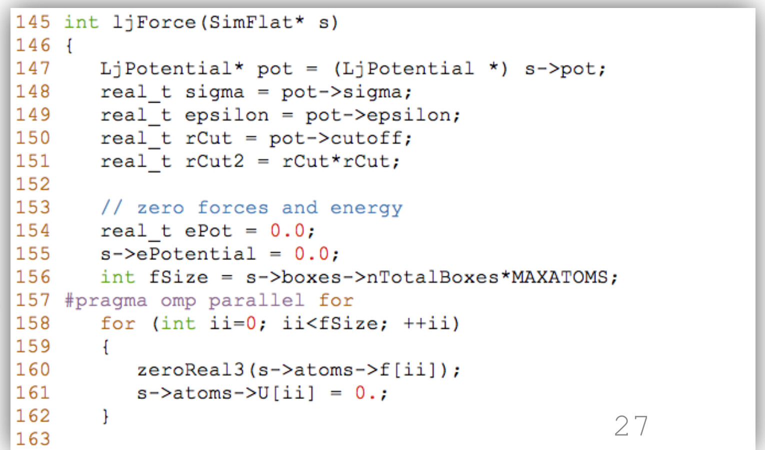
```
LOOP BEGIN at ljForce.c(157,1)
   remark #15344: loop was not vectorized: vector dependence prevents vectorization
   remark #15346: vector dependence: assumed OUTPUT dependence between s->atoms->f[ii] (22:4) and s->atoms->U[ii] (162:7)
   remark #15346: vector dependence: assumed OUTPUT dependence between s->atoms->U[ii] (162:7) and s->atoms->f[ii] (22:4)
   remark #25439: unrolled with remainder by 2
LOOP END

LOOP BEGIN at ljForce.c(157,1)
<Remainder>
LOOP END

```

Optimization Report Summary:

Initialization Step of Baseline Code:

1. No vectorization
2. Function call in loop:
   - Line: 160
   - Function: ”zeroReal3”
3. Remainder loop
4. Assumed dependence

Source Code:



```
145 int ljForce(SimFlat* s)
146 {
147    LjPotential* pot = (LjPotential *) s->pot;
148    real_t sigma = pot->sigma;
149    real_t epsilon = pot->epsilon;
150    real_t rCut = pot->cutoff;
151    real_t rCut2 = rCut*rCut;
152
153    // zero forces and energy
154    real_t ePot = 0.0;
155    s->ePotential = 0.0;
156    int fSize = s->boxes->nTotalBoxes*MAXATOMS;
157 #pragma omp parallel for
158    for (int ii=0; ii<fSize; ++ii)
159    {
160       zeroReal3(s->atoms->f[ii]);
161       s->atoms->U[ii] = 0.;
162    }
163
```

L-J Force:

Summary of Fix List:

1. No vectorization
2. Remainder loop
3. Assumed dependence

Fixing Item 1 - "No vectorization":

Add:

"#pragma omp simd"

Or Add:

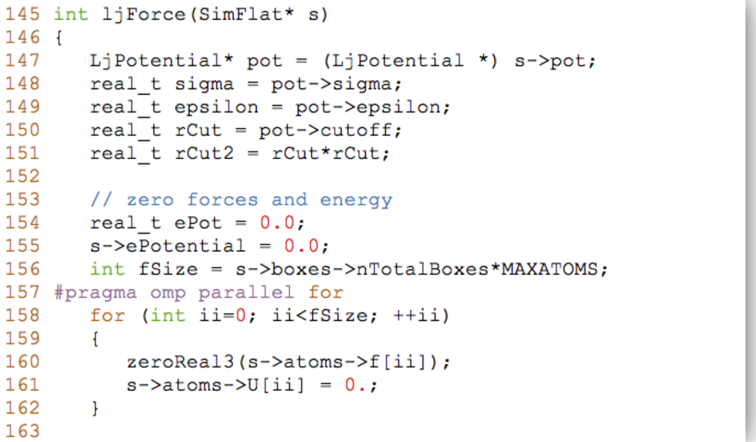
"#pragma omp parallel for simd"

Why Not?:

"#pragma simd"

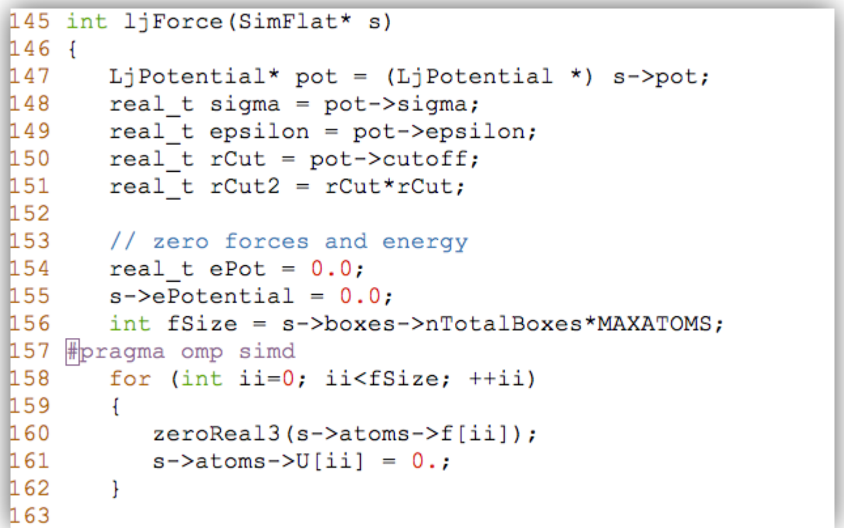
L-J Force:

Source Code Before:



```
int ljForce(SimFlat* s)
{
   LjPotential* pot = (LjPotential *) s->pot;
   real_t sigma = pot->sigma;
   real_t epsilon = pot->epsilon;
   real_t rCut = pot->cutoff;
   real_t rCut2 = rCut*rCut;

   // zero forces and energy
   real_t ePot = 0.0;
   s->ePotential = 0.0;
   int fSize = s->boxes->nTotalBoxes*MAXATOMS;
#pragma omp parallel for
   for (int ii=0; ii<fSize; ++ii)
   {
      zeroReal3(s->atoms->f[ii]);
      s->atoms->U[ii] = 0.;
   }

```

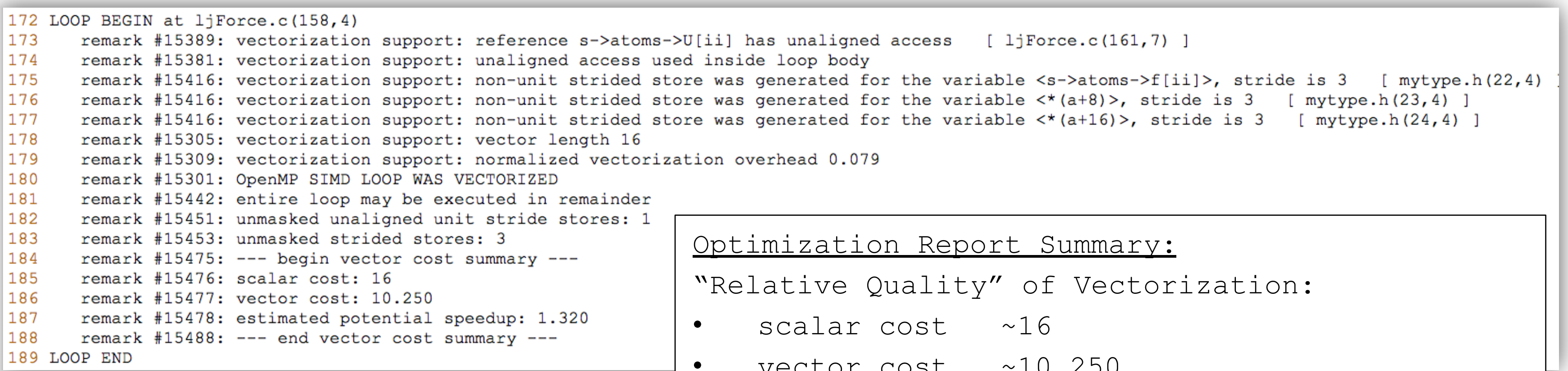
Source Code After:



```
int ljForce(SimFlat* s)
{
   LjPotential* pot = (LjPotential *) s->pot;
   real_t sigma = pot->sigma;
   real_t epsilon = pot->epsilon;
   real_t rCut = pot->cutoff;
   real_t rCut2 = rCut*rCut;

   // zero forces and energy
   real_t ePot = 0.0;
   s->ePotential = 0.0;
   int fSize = s->boxes->nTotalBoxes*MAXATOMS;
#pragma omp simd
   for (int ii=0; ii<fSize; ++ii)
   {
      zeroReal3(s->atoms->f[ii]);
      s->atoms->U[ii] = 0.;
   }

```

L-J Force:

Optimization Reports:



```
LOOP BEGIN at ljForce.c(158,4)
   remark #15389: vectorization support: reference s->atoms->U[ii] has unaligned access   [ ljForce.c(161,7) ]
   remark #15381: vectorization support: unaligned access used inside loop body
   remark #15416: vectorization support: non-unit strided store was generated for the variable <s->atoms->f[ii]>, stride is 3   [ mytype.h(22,4) ]
   remark #15416: vectorization support: non-unit strided store was generated for the variable <*(a+8)>, stride is 3   [ mytype.h(23,4) ]
   remark #15416: vectorization support: non-unit strided store was generated for the variable <*(a+16)>, stride is 3   [ mytype.h(24,4) ]
   remark #15305: vectorization support: vector length 16
   remark #15309: vectorization support: normalized vectorization overhead 0.079
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15451: unmasked unaligned unit stride stores: 1
   remark #15453: unmasked strided stores: 3
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 16
   remark #15477: vector cost: 10.250
   remark #15478: estimated potential speedup: 1.320
   remark #15488: --- end vector cost summary ---
LOOP END
```

Optimization Report Summary:

"Relative Quality" of Vectorization:

- scalar cost ~16
- vector cost ~10.250
- Est. Speed-Up ~1.320 (low)

Necessary fixes:

1. Non-SIMD-Enabled function – "zeroReal3"
2. Unaligned access:
   - "s->atom->U" structure
   - "s->atom->f" structure
3. Non-unit (3) stride store: "s->atom->f"

L-J Force:

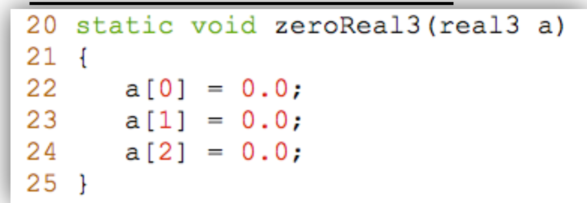
Fixing Item 1 - "Function call in loop":

- Convert "zeroReal3" function:
  - SIMD-Enabled function
    - Part 1: Declaration
    - Part 2: Definition

OR:

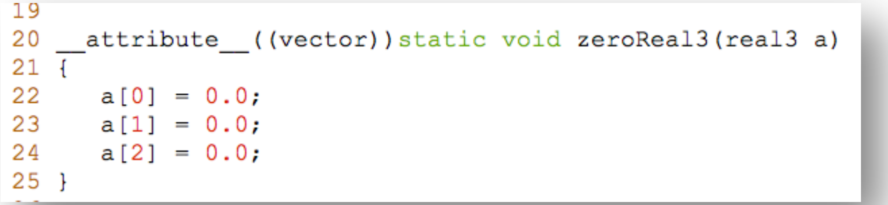
- SIMD-Enabled omp function
  - Part 1: Declaration
  - Part 2: Definition:
    - Add "notinbranch" clause
    - "omp declare simd" directive

Source Code Before:



```
static void zeroReal3(real3 a)
{
    a[0] = 0.0;
    a[1] = 0.0;
    a[2] = 0.0;
}
```

Source Code After:



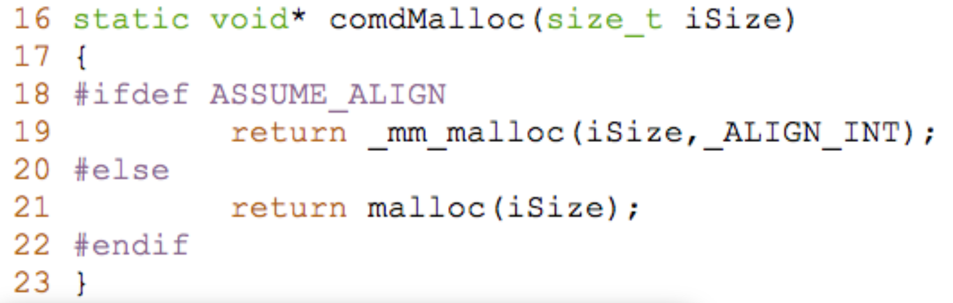
```

__attribute__((vector))static void zeroReal3(real3 a)
{
    a[0] = 0.0;
    a[1] = 0.0;
    a[2] = 0.0;
}
```

L-J Force:

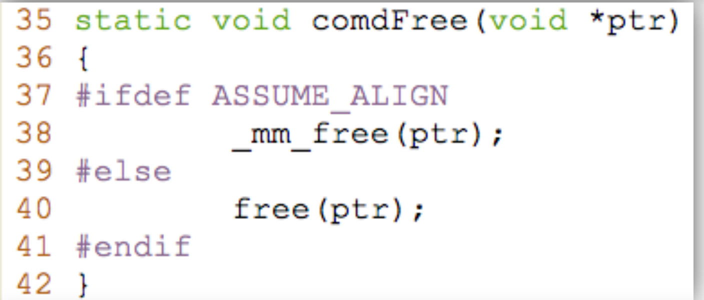
Fixing Item 2 - “Unaligned access”:

- Unaligned access:
  - “s->atom->U” structure
  - “s->atom->f” structure
- How to Align:
  - Part 1: “_mm_malloc”
  - Part 2: “_mm_free”
  - Part 3: “inform compiler”
    - Before use

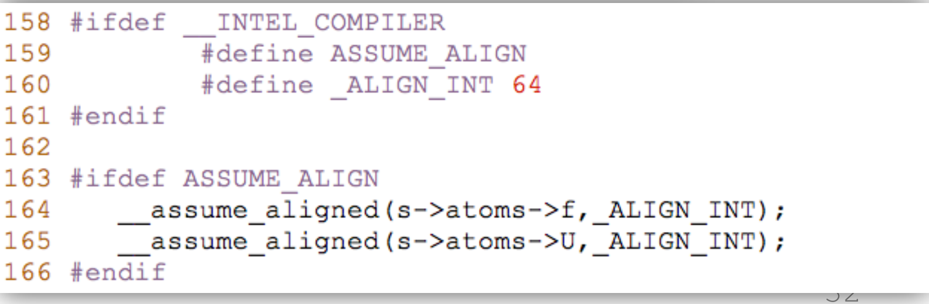
Aligned Allocation & Free: “memUtils.h”:



```
static void* comdMalloc(size_t iSize)
{
#ifdef ASSUME_ALIGN
        return _mm_malloc(iSize,_ALIGN_INT);
#else
        return malloc(iSize);
#endif
}
```



```
static void comdFree(void *ptr)
{
#ifdef ASSUME_ALIGN
        _mm_free(ptr);
#else
        free(ptr);
#endif
}
```



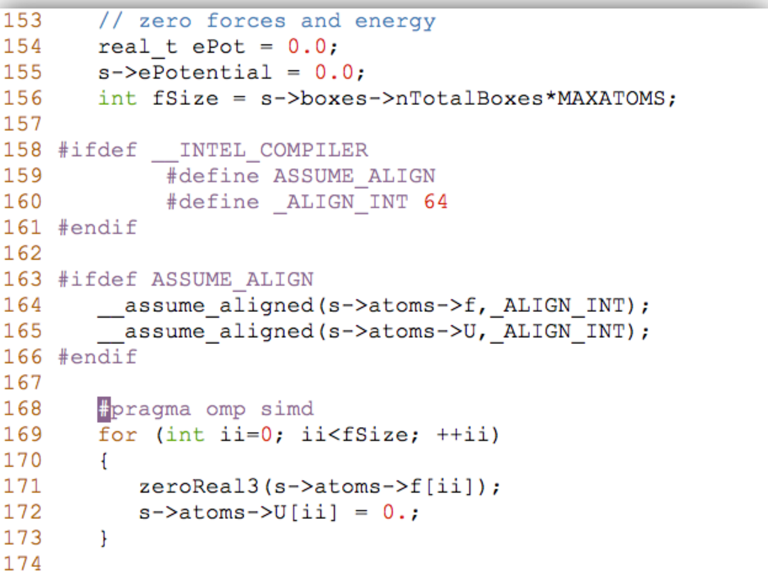
```
#ifdef __INTEL_COMPILER
        #define ASSUME_ALIGN
        #define _ALIGN_INT 64
#endif

#ifdef ASSUME_ALIGN
    __assume_aligned(s->atoms->f,_ALIGN_INT);
    __assume_aligned(s->atoms->U,_ALIGN_INT);
#endif
```

L-j Force:

Fixing Item 3 - "Non-unit stride store - 3":

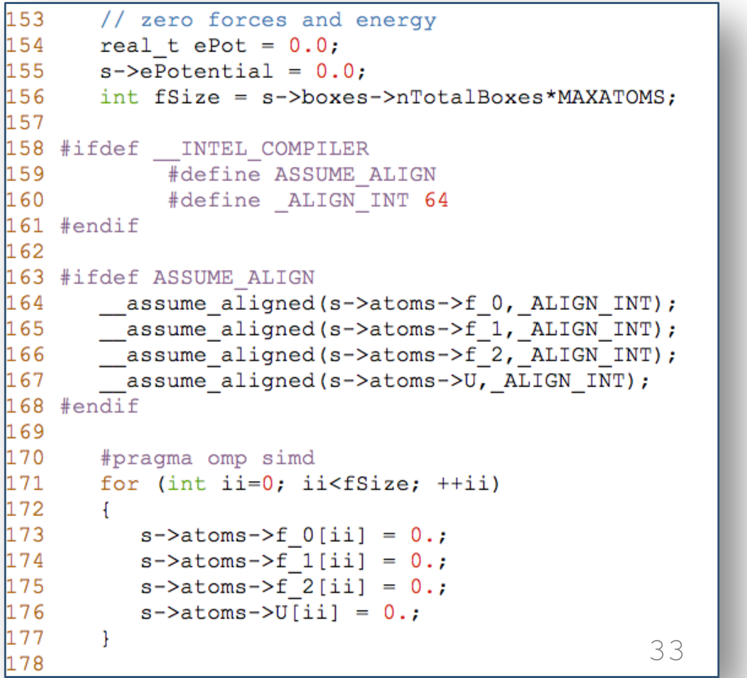
- Convert So(AoA) to:
    - SoA_0, SoA_1, SoA_2

Source Code Before:



```
   // zero forces and energy
   real_t ePot = 0.0;
   s->ePotential = 0.0;
   int fSize = s->boxes->nTotalBoxes*MAXATOMS;

#ifdef __INTEL_COMPILER
        #define ASSUME_ALIGN
        #define _ALIGN_INT 64
#endif

#ifdef ASSUME_ALIGN
   __assume_aligned(s->atoms->f,_ALIGN_INT);
   __assume_aligned(s->atoms->U,_ALIGN_INT);
#endif

   #pragma omp simd
   for (int ii=0; ii<fSize; ++ii)
   {
      zeroReal3(s->atoms->f[ii]);
      s->atoms->U[ii] = 0.;
   }

```

Source Code After:

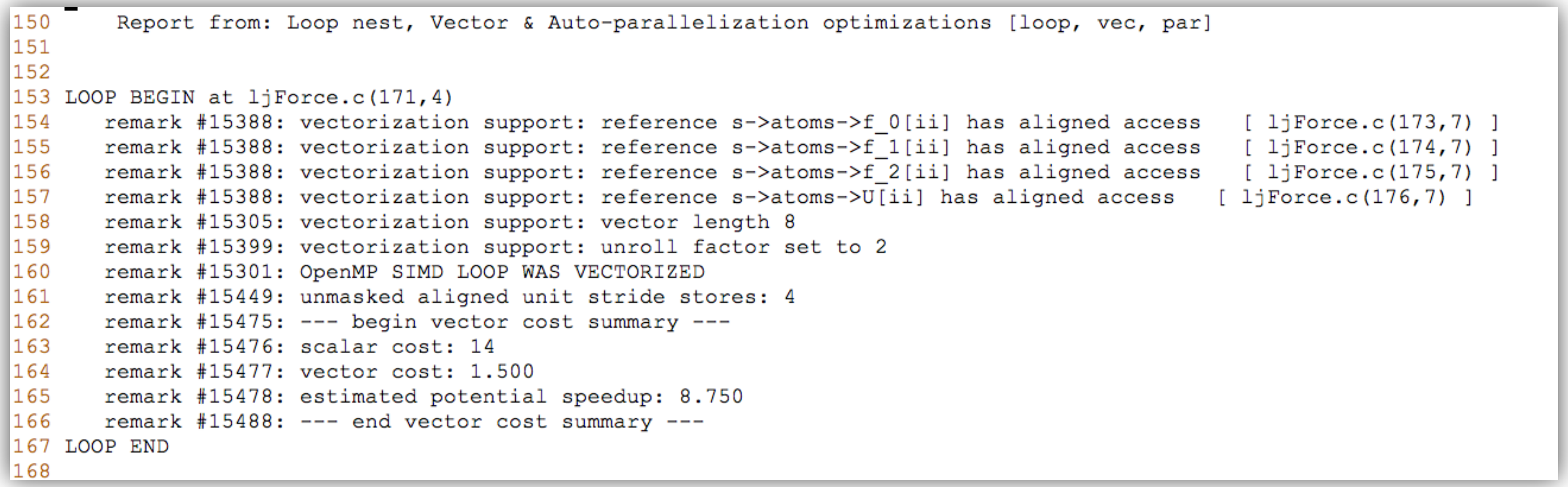


```
   // zero forces and energy
   real_t ePot = 0.0;
   s->ePotential = 0.0;
   int fSize = s->boxes->nTotalBoxes*MAXATOMS;

#ifdef __INTEL_COMPILER
        #define ASSUME_ALIGN
        #define _ALIGN_INT 64
#endif

#ifdef ASSUME_ALIGN
   __assume_aligned(s->atoms->f_0,_ALIGN_INT);
   __assume_aligned(s->atoms->f_1,_ALIGN_INT);
   __assume_aligned(s->atoms->f_2,_ALIGN_INT);
   __assume_aligned(s->atoms->U,_ALIGN_INT);
#endif

   #pragma omp simd
   for (int ii=0; ii<fSize; ++ii)
   {
      s->atoms->f_0[ii] = 0.;
      s->atoms->f_1[ii] = 0.;
      s->atoms->f_2[ii] = 0.;
      s->atoms->U[ii] = 0.;
   }

```

L-J Force:

Final Optimization Report:

1. All Aligned Access
2. No Peel/Remainder Loops
3. Relative Vectorization Quality (up ~90%)



```
    Report from: Loop nest, Vector & Auto-parallelization optimizations [loop, vec, par]


LOOP BEGIN at ljForce.c(171,4)
   remark #15388: vectorization support: reference s->atoms->f_0[ii] has aligned access   [ ljForce.c(173,7) ]
   remark #15388: vectorization support: reference s->atoms->f_1[ii] has aligned access   [ ljForce.c(174,7) ]
   remark #15388: vectorization support: reference s->atoms->f_2[ii] has aligned access   [ ljForce.c(175,7) ]
   remark #15388: vectorization support: reference s->atoms->U[ii] has aligned access   [ ljForce.c(176,7) ]
   remark #15305: vectorization support: vector length 8
   remark #15399: vectorization support: unroll factor set to 2
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   remark #15449: unmasked aligned unit stride stores: 4
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 14
   remark #15477: vector cost: 1.500
   remark #15478: estimated potential speedup: 8.750
   remark #15488: --- end vector cost summary ---
LOOP END
```

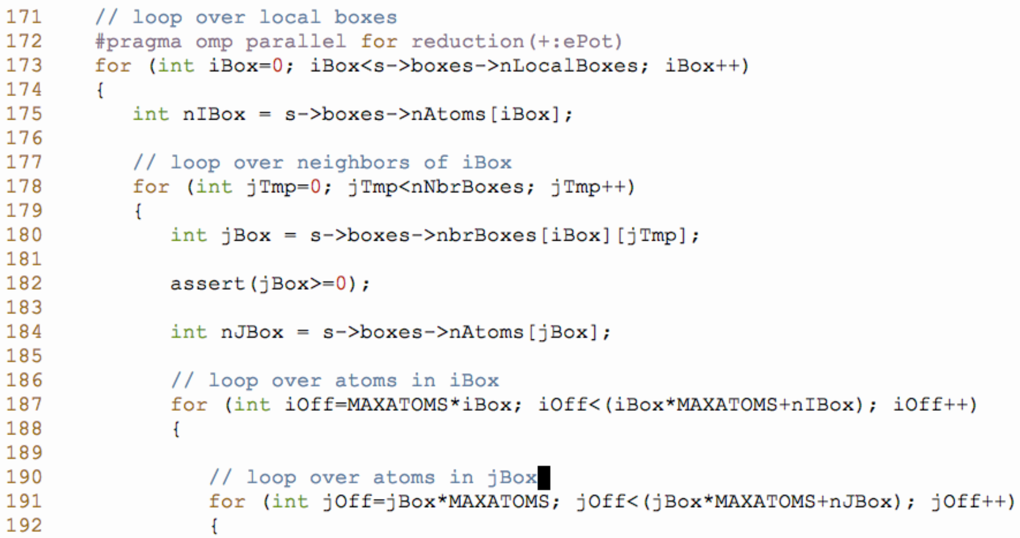
Source Code: CoMD

L-J Force Kernel

Force Calculation Step

L-J Force:

Showing 4 Nested Loops:



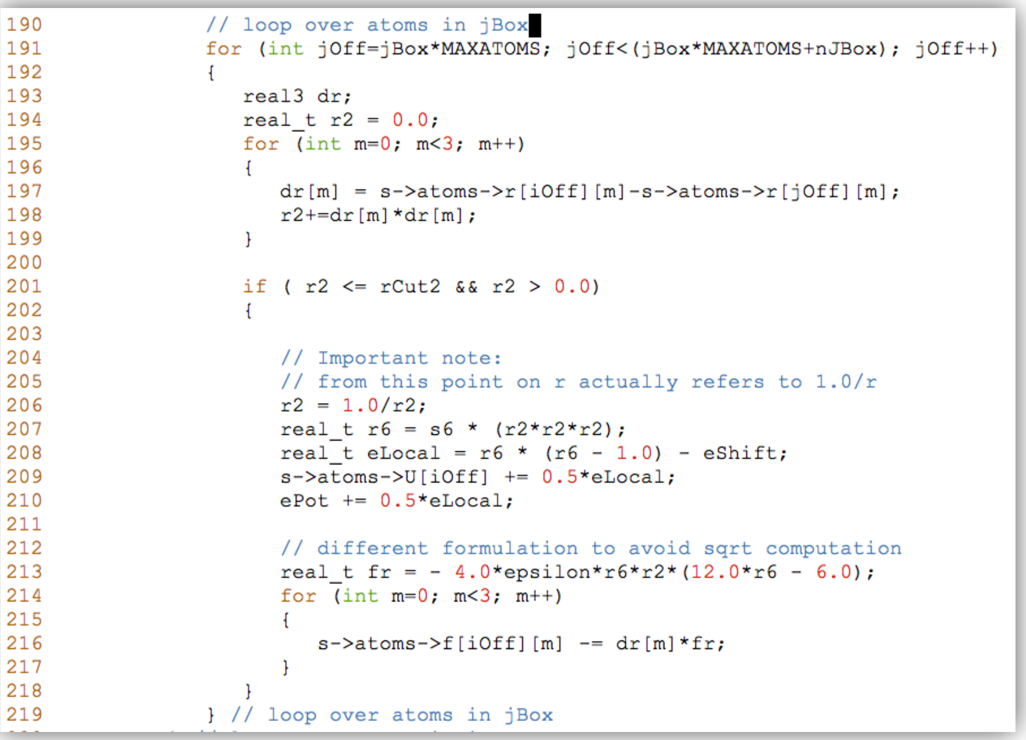
```
171    // loop over local boxes
172    #pragma omp parallel for reduction(+:ePot)
173    for (int iBox=0; iBox<s->boxes->nLocalBoxes; iBox++)
174    {
175       int nIBox = s->boxes->nAtoms[iBox];
176
177       // loop over neighbors of iBox
178       for (int jTmp=0; jTmp<nNbrBoxes; jTmp++)
179       {
180          int jBox = s->boxes->nbrBoxes[iBox][jTmp];
181
182          assert(jBox>=0);
183
184          int nJBox = s->boxes->nAtoms[jBox];
185
186          // loop over atoms in iBox
187          for (int iOff=MAXATOMS*iBox; iOff<(iBox*MAXATOMS+nIBox); iOff++)
188          {
189
190             // loop over atoms in jBox
191             for (int jOff=jBox*MAXATOMS; jOff<(jBox*MAXATOMS+nJBox); jOff++)
192             {
```

Source Code Summary:

- Force Calculation Step:
  - 4 Nested Loops
  - “omp reduction” on:
    - “ePot”
  - Imperfect nested loops
    - Loop @Line 195
    - Loop @Line 180-182
  - 2 perfectly nested loops:
    - Loop @Line 187
    - Loop @Line 191

L-J Force:

Innermost Loop :



```
190    // loop over atoms in jBox
191    for (int jOff=jBox*MAXATOMS; jOff<(jBox*MAXATOMS+nJBox); jOff++)
192    {
193       real3 dr;
194       real_t r2 = 0.0;
195       for (int m=0; m<3; m++)
196       {
197          dr[m] = s->atoms->r[iOff][m]-s->atoms->r[jOff][m];
198          r2+=dr[m]*dr[m];
199       }
200
201       if ( r2 <= rCut2 && r2 > 0.0)
202       {
203
204          // Important note:
205          // from this point on r actually refers to 1.0/r
206          r2 = 1.0/r2;
207          real_t r6 = s6 * (r2*r2*r2);
208          real_t eLocal = r6 * (r6 - 1.0) - eShift;
209          s->atoms->U[iOff] += 0.5*eLocal;
210          ePot += 0.5*eLocal;
211
212          // different formulation to avoid sqrt computation
213          real_t fr = - 4.0*epsilon*r6*r2*(12.0*r6 - 6.0);
214          for (int m=0; m<3; m++)
215          {
216             s->atoms->f[iOff][m] -= dr[m]*fr;
217          }
218       }
219    } // loop over atoms in jBox
```

Source Code Summary:

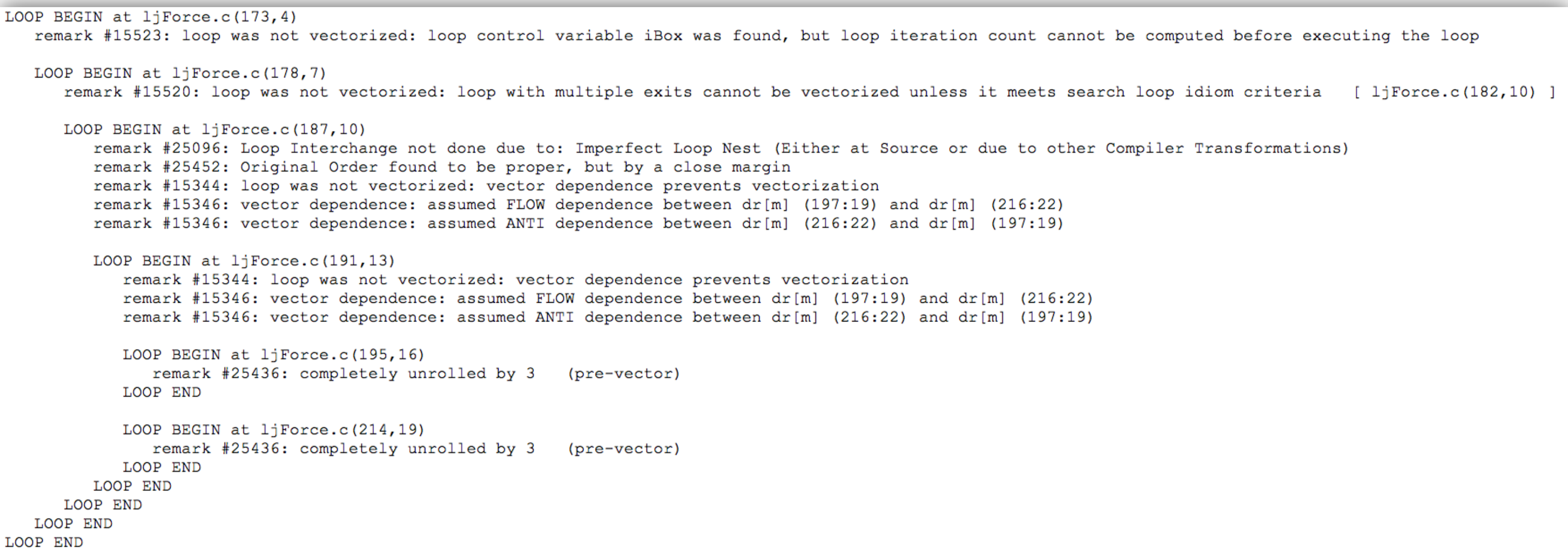
- Force Calculation Step:
  - Inner most loop constains:
    - Sub-Loops @Line 195
    - Sub-Loops @Line 214
  - Omp reduction on variable
    - "ePot"

L-J Force:

Optimization Report Summary:

1. No vectorization:
   - 4 Nested-Loops
2. Imperfect loop nest
3. Assumed flow dependence:
   - Loop @187
   - Loop @191

Optimization Reports:

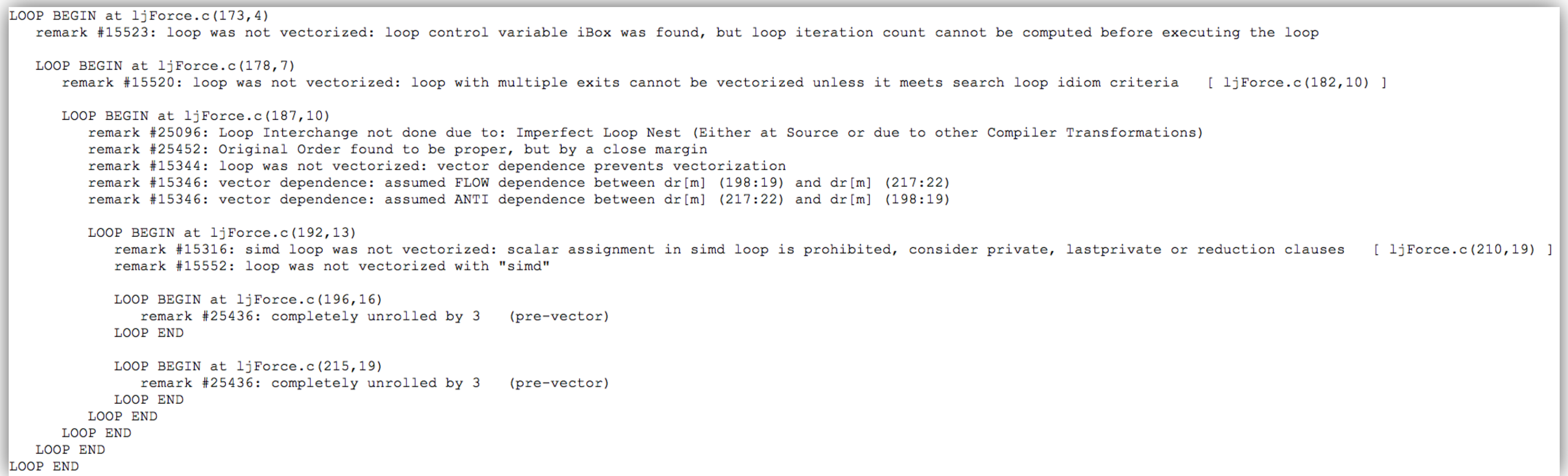


```
LOOP BEGIN at ljForce.c(173,4)
   remark #15523: loop was not vectorized: loop control variable iBox was found, but loop iteration count cannot be computed before executing the loop

   LOOP BEGIN at ljForce.c(178,7)
      remark #15520: loop was not vectorized: loop with multiple exits cannot be vectorized unless it meets search loop idiom criteria   [ ljForce.c(182,10) ]

      LOOP BEGIN at ljForce.c(187,10)
         remark #25096: Loop Interchange not done due to: Imperfect Loop Nest (Either at Source or due to other Compiler Transformations)
         remark #25452: Original Order found to be proper, but by a close margin
         remark #15344: loop was not vectorized: vector dependence prevents vectorization
         remark #15346: vector dependence: assumed FLOW dependence between dr[m] (197:19) and dr[m] (216:22)
         remark #15346: vector dependence: assumed ANTI dependence between dr[m] (216:22) and dr[m] (197:19)

         LOOP BEGIN at ljForce.c(191,13)
            remark #15344: loop was not vectorized: vector dependence prevents vectorization
            remark #15346: vector dependence: assumed FLOW dependence between dr[m] (197:19) and dr[m] (216:22)
            remark #15346: vector dependence: assumed ANTI dependence between dr[m] (216:22) and dr[m] (197:19)

            LOOP BEGIN at ljForce.c(195,16)
               remark #25436: completely unrolled by 3   (pre-vector)
            LOOP END

            LOOP BEGIN at ljForce.c(214,19)
               remark #25436: completely unrolled by 3   (pre-vector)
            LOOP END
         LOOP END
      LOOP END
   LOOP END
LOOP END
```

L-J Force:

1. Fixing Item 1 – "No vectorization":
   - Add "simd" directive to innermost loop
     - Sub-Loops: 196, 215 get unrolled
   - Note:
     - There are 4 nested loops

Optimization Reports:



```
LOOP BEGIN at ljForce.c(173,4)
   remark #15523: loop was not vectorized: loop control variable iBox was found, but loop iteration count cannot be computed before executing the loop

   LOOP BEGIN at ljForce.c(178,7)
      remark #15520: loop was not vectorized: loop with multiple exits cannot be vectorized unless it meets search loop idiom criteria   [ ljForce.c(182,10) ]

      LOOP BEGIN at ljForce.c(187,10)
         remark #25096: Loop Interchange not done due to: Imperfect Loop Nest (Either at Source or due to other Compiler Transformations)
         remark #25452: Original Order found to be proper, but by a close margin
         remark #15344: loop was not vectorized: vector dependence prevents vectorization
         remark #15346: vector dependence: assumed FLOW dependence between dr[m] (198:19) and dr[m] (217:22)
         remark #15346: vector dependence: assumed ANTI dependence between dr[m] (217:22) and dr[m] (198:19)

         LOOP BEGIN at ljForce.c(192,13)
            remark #15316: simd loop was not vectorized: scalar assignment in simd loop is prohibited, consider private, lastprivate or reduction clauses   [ ljForce.c(210,19) ]
            remark #15552: loop was not vectorized with "simd"

            LOOP BEGIN at ljForce.c(196,16)
               remark #25436: completely unrolled by 3   (pre-vector)
            LOOP END

            LOOP BEGIN at ljForce.c(215,19)
               remark #25436: completely unrolled by 3   (pre-vector)
            LOOP END
         LOOP END
      LOOP END
   LOOP END
LOOP END
```

L-J Force:

Summary of Todo:

1. Align Data & Hint Compiler:
    - “s->atoms->r”
    - “s->atoms->f” …done!

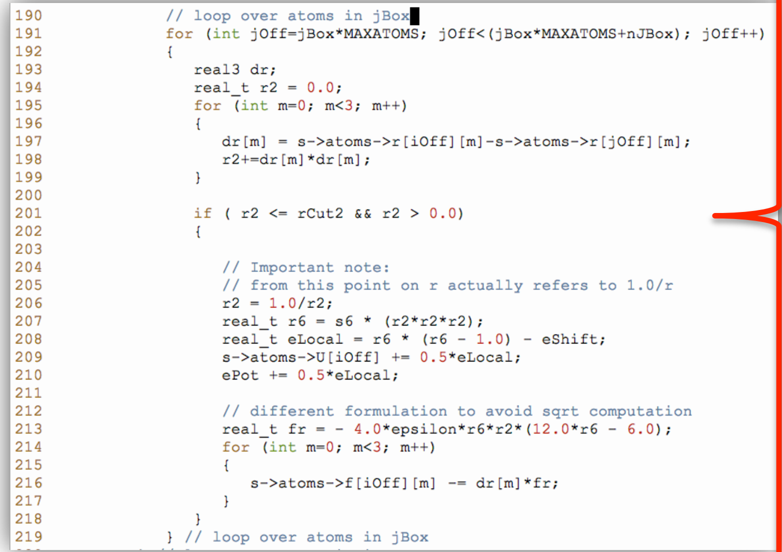
2. Convert So(AoA) to multiple SoA:

    “s->atoms->r”
    “s->atoms->f” …done!

3. Convert reduction in inner loop on:
    - “s->atoms->r” to “simd reduction”
    - “s->atoms->f” to “simd reduction”
    - Hint:
        - Introduce summation variable
            - “sum_R”
            - “sum_F”

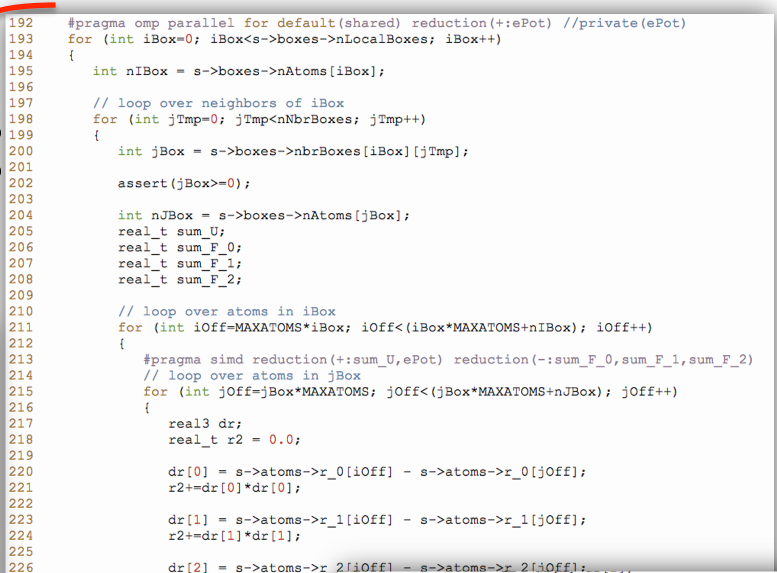
L-J Force:

Source Code Before:

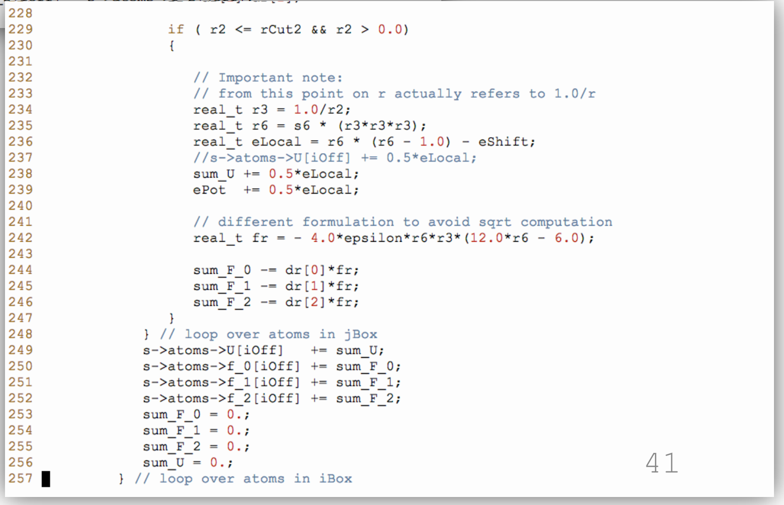


```
// loop over atoms in jBox
for (int jOff=jBox*MAXATOMS; jOff<(jBox*MAXATOMS+nJBox); jOff++)
{
   real3 dr;
   real_t r2 = 0.0;
   for (int m=0; m<3; m++)
   {
      dr[m] = s->atoms->r[iOff][m]-s->atoms->r[jOff][m];
      r2+=dr[m]*dr[m];
   }

   if ( r2 <= rCut2 && r2 > 0.0)
   {

      // Important note:
      // from this point on r actually refers to 1.0/r
      r2 = 1.0/r2;
      real_t r6 = s6 * (r2*r2*r2);
      real_t eLocal = r6 * (r6 - 1.0) - eShift;
      s->atoms->U[iOff] += 0.5*eLocal;
      ePot += 0.5*eLocal;

      // different formulation to avoid sqrt computation
      real_t fr = - 4.0*epsilon*r6*r2*(12.0*r6 - 6.0);
      for (int m=0; m<3; m++)
      {
         s->atoms->f[iOff][m] -= dr[m]*fr;
      }
   }
} // loop over atoms in jBox
```

Source Code After:



```
#pragma omp parallel for default(shared) reduction(+:ePot) //private(ePot)
for (int iBox=0; iBox<s->boxes->nLocalBoxes; iBox++)
{
   int nIBox = s->boxes->nAtoms[iBox];

   // loop over neighbors of iBox
   for (int jTmp=0; jTmp<nNbrBoxes; jTmp++)
   {
      int jBox = s->boxes->nbrBoxes[iBox][jTmp];

      assert(jBox>=0);

      int nJBox = s->boxes->nAtoms[jBox];
      real_t sum_U;
      real_t sum_F_0;
      real_t sum_F_1;
      real_t sum_F_2;

      // loop over atoms in iBox
      for (int iOff=MAXATOMS*iBox; iOff<(iBox*MAXATOMS+nIBox); iOff++)
      {
         #pragma simd reduction(+:sum_U,ePot) reduction(-:sum_F_0,sum_F_1,sum_F_2)
         // loop over atoms in jBox
         for (int jOff=jBox*MAXATOMS; jOff<(jBox*MAXATOMS+nJBox); jOff++)
         {
            real3 dr;
            real_t r2 = 0.0;

            dr[0] = s->atoms->r_0[iOff] - s->atoms->r_0[jOff];
            r2+=dr[0]*dr[0];

            dr[1] = s->atoms->r_1[iOff] - s->atoms->r_1[jOff];
            r2+=dr[1]*dr[1];

            dr[2] = s->atoms->r_2[iOff] - s->atoms->r_2[jOff];
            r2+=dr[2]*dr[2];
```



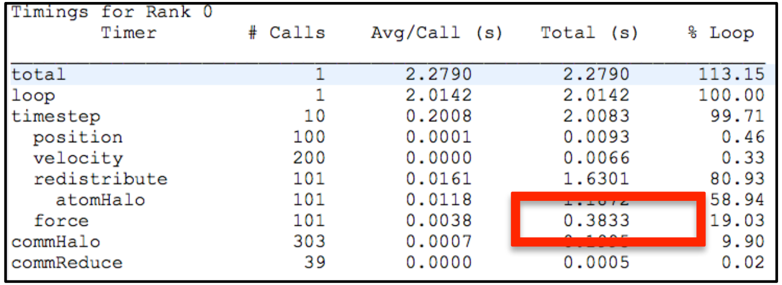
```

            if ( r2 <= rCut2 && r2 > 0.0)
            {

               // Important note:
               // from this point on r actually refers to 1.0/r
               real_t r3 = 1.0/r2;
               real_t r6 = s6 * (r3*r3*r3);
               real_t eLocal = r6 * (r6 - 1.0) - eShift;
               //s->atoms->U[iOff] += 0.5*eLocal;
               sum_U += 0.5*eLocal;
               ePot  += 0.5*eLocal;

               // different formulation to avoid sqrt computation
               real_t fr = - 4.0*epsilon*r6*r3*(12.0*r6 - 6.0);

               sum_F_0 -= dr[0]*fr;
               sum_F_1 -= dr[1]*fr;
               sum_F_2 -= dr[2]*fr;
            }
         } // loop over atoms in jBox
         s->atoms->U[iOff]    += sum_U;
         s->atoms->f_0[iOff] += sum_F_0;
         s->atoms->f_1[iOff] += sum_F_1;
         s->atoms->f_2[iOff] += sum_F_2;
         sum_F_0 = 0.;
         sum_F_1 = 0.;
         sum_F_2 = 0.;
         sum_U = 0.;
      } // loop over atoms in iBox
```

L-J Force – Optimized

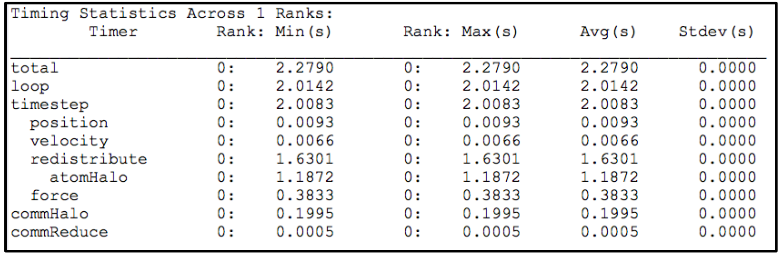
Performance Report”:

- ~40% Speedup in Force Calculation
- ~10% Speedup overall

Old Output:

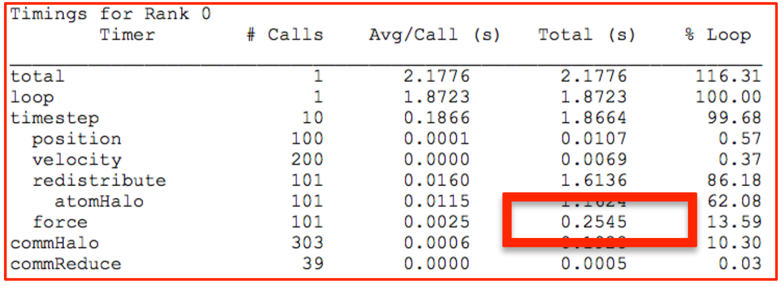


```
Timings for Rank 0
        Timer          # Calls    Avg/Call (s)    Total (s)    % Loop
_____________________________________________________________________
total                        1        2.2790        2.2790     113.15
loop                         1        2.0142        2.0142     100.00
timestep                    10        0.2008        2.0083      99.71
  position                 100        0.0001        0.0093       0.46
  velocity                 200        0.0000        0.0066       0.33
  redistribute             101        0.0161        1.6301      80.93
    atomHalo               101        0.0118        1.1872      58.94
  force                    101        0.0038        0.3833      19.03
commHalo                   303        0.0007        0.1995       9.90
commReduce                  39        0.0000        0.0005       0.02
```

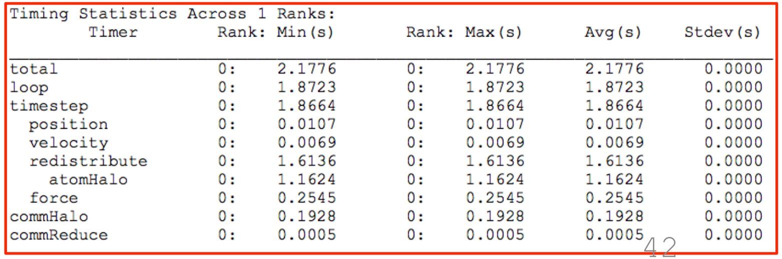


```
Timing Statistics Across 1 Ranks:
       Timer         Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total                 0:   2.2790        0:   2.2790      2.2790      0.0000
loop                  0:   2.0142        0:   2.0142      2.0142      0.0000
timestep              0:   2.0083        0:   2.0083      2.0083      0.0000
  position            0:   0.0093        0:   0.0093      0.0093      0.0000
  velocity            0:   0.0066        0:   0.0066      0.0066      0.0000
  redistribute        0:   1.6301        0:   1.6301      1.6301      0.0000
    atomHalo          0:   1.1872        0:   1.1872      1.1872      0.0000
  force               0:   0.3833        0:   0.3833      0.3833      0.0000
commHalo              0:   0.1995        0:   0.1995      0.1995      0.0000
commReduce            0:   0.0005        0:   0.0005      0.0005      0.0000
```

New Output:



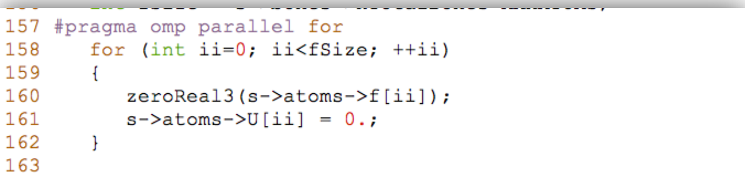
```
Timings for Rank 0
        Timer          # Calls    Avg/Call (s)    Total (s)    % Loop
_____________________________________________________________________
total                        1        2.1776        2.1776     116.31
loop                         1        1.8723        1.8723     100.00
timestep                    10        0.1866        1.8664      99.68
  position                 100        0.0001        0.0107       0.57
  velocity                 200        0.0000        0.0069       0.37
  redistribute             101        0.0160        1.6136      86.18
    atomHalo               101        0.0115        1.1624      62.08
  force                    101        0.0025        0.2545      13.59
commHalo                   303        0.0006        0.1928      10.30
commReduce                  39        0.0000        0.0005       0.03
```



```
Timing Statistics Across 1 Ranks:
       Timer         Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total                 0:   2.1776        0:   2.1776      2.1776      0.0000
loop                  0:   1.8723        0:   1.8723      1.8723      0.0000
timestep              0:   1.8664        0:   1.8664      1.8664      0.0000
  position            0:   0.0107        0:   0.0107      0.0107      0.0000
  velocity            0:   0.0069        0:   0.0069      0.0069      0.0000
  redistribute        0:   1.6136        0:   1.6136      1.6136      0.0000
    atomHalo          0:   1.1624        0:   1.1624      1.1624      0.0000
  force               0:   0.2545        0:   0.2545      0.2545      0.0000
commHalo              0:   0.1928        0:   0.1928      0.1928      0.0000
commReduce            0:   0.0005        0:   0.0005      0.0005      0.0000
```

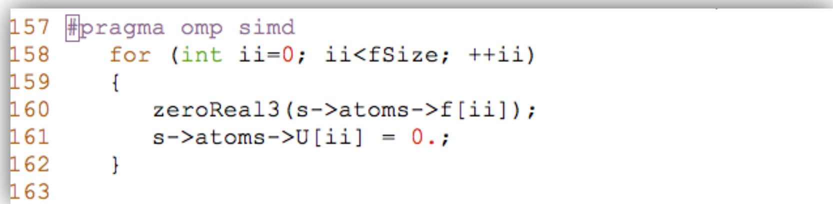
L-J Force – Revision:

Source Code Before:



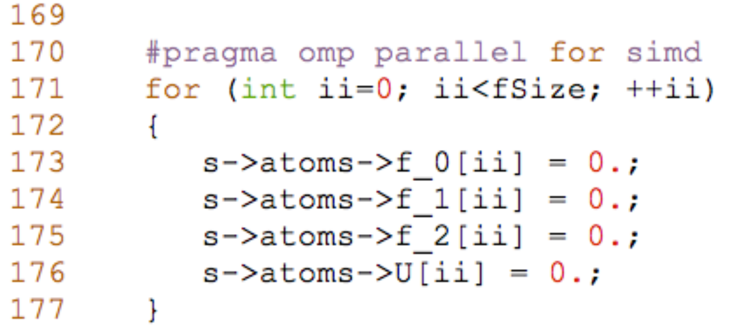
```
#pragma omp parallel for
    for (int ii=0; ii<fSize; ++ii)
    {
        zeroReal3(s->atoms->f[ii]);
        s->atoms->U[ii] = 0.;
    }
```

Source Code After Revision I:



```
#pragma omp simd
    for (int ii=0; ii<fSize; ++ii)
    {
        zeroReal3(s->atoms->f[ii]);
        s->atoms->U[ii] = 0.;
    }
```

Source Code After Revision II:



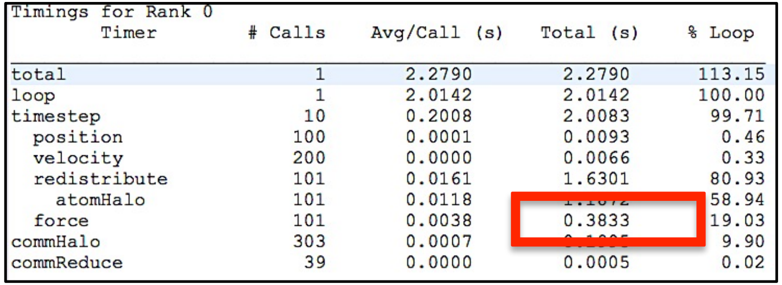
```

    #pragma omp parallel for simd
    for (int ii=0; ii<fSize; ++ii)
    {
        s->atoms->f_0[ii] = 0.;
        s->atoms->f_1[ii] = 0.;
        s->atoms->f_2[ii] = 0.;
        s->atoms->U[ii] = 0.;
    }
```

L-J Force – Optimized

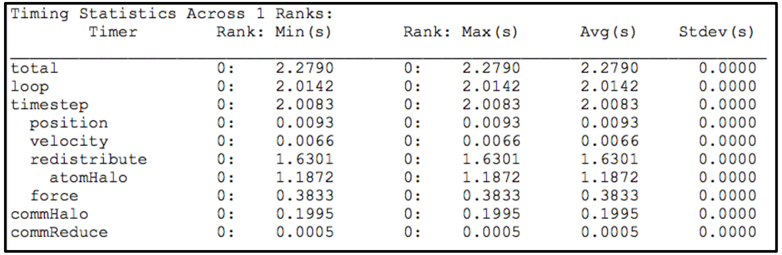
Revised Performance Report”:

- Revised Initialization Step
- ~51% Speedup in Force Calculation
- ~10% Speedup overall

Old Output:

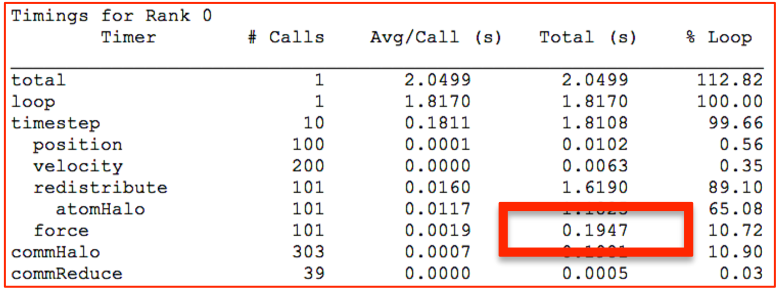


```
Timings for Rank 0
          Timer           # Calls     Avg/Call (s)    Total (s)     % Loop

total                           1         2.2790        2.2790      113.15
loop                            1         2.0142        2.0142      100.00
timestep                       10         0.2008        2.0083       99.71
  position                    100         0.0001        0.0093        0.46
  velocity                    200         0.0000        0.0066        0.33
  redistribute                101         0.0161        1.6301       80.93
    atomHalo                  101         0.0118        1.1872       58.94
  force                       101         0.0038        0.3833       19.03
commHalo                      303         0.0007        0.1995        9.90
commReduce                     39         0.0000        0.0005        0.02
```

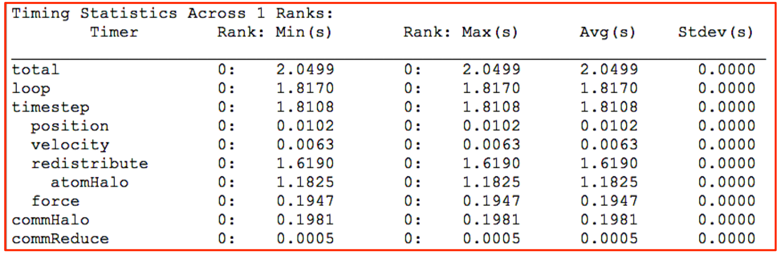


```
Timing Statistics Across 1 Ranks:
        Timer         Rank: Min(s)         Rank: Max(s)         Avg(s)      Stdev(s)

total                 0:    2.2790         0:    2.2790         2.2790      0.0000
loop                  0:    2.0142         0:    2.0142         2.0142      0.0000
timestep              0:    2.0083         0:    2.0083         2.0083      0.0000
  position            0:    0.0093         0:    0.0093         0.0093      0.0000
  velocity            0:    0.0066         0:    0.0066         0.0066      0.0000
  redistribute        0:    1.6301         0:    1.6301         1.6301      0.0000
    atomHalo          0:    1.1872         0:    1.1872         1.1872      0.0000
  force               0:    0.3833         0:    0.3833         0.3833      0.0000
commHalo              0:    0.1995         0:    0.1995         0.1995      0.0000
commReduce            0:    0.0005         0:    0.0005         0.0005      0.0000
```

New Output:



```
Timings for Rank 0
          Timer           # Calls     Avg/Call (s)    Total (s)     % Loop

total                           1         2.0499        2.0499      112.82
loop                            1         1.8170        1.8170      100.00
timestep                       10         0.1811        1.8108       99.66
  position                    100         0.0001        0.0102        0.56
  velocity                    200         0.0000        0.0063        0.35
  redistribute                101         0.0160        1.6190       89.10
    atomHalo                  101         0.0117        1.1825       65.08
  force                       101         0.0019        0.1947       10.72
commHalo                      303         0.0007        0.1981       10.90
commReduce                     39         0.0000        0.0005        0.03
```



```
Timing Statistics Across 1 Ranks:
        Timer         Rank: Min(s)         Rank: Max(s)         Avg(s)      Stdev(s)

total                 0:    2.0499         0:    2.0499         2.0499      0.0000
loop                  0:    1.8170         0:    1.8170         1.8170      0.0000
timestep              0:    1.8108         0:    1.8108         1.8108      0.0000
  position            0:    0.0102         0:    0.0102         0.0102      0.0000
  velocity            0:    0.0063         0:    0.0063         0.0063      0.0000
  redistribute        0:    1.6190         0:    1.6190         1.6190      0.0000
    atomHalo          0:    1.1825         0:    1.1825         1.1825      0.0000
  force               0:    0.1947         0:    0.1947         0.1947      0.0000
commHalo              0:    0.1981         0:    0.1981         0.1981      0.0000
commReduce            0:    0.0005         0:    0.0005         0.0005      0.0000
```

Source Code: CoMD

EAM-Force Kernel

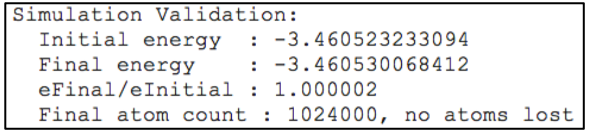
Force Calculation Step

EAM Force – Baseline Simulation Output:

Problem Definitions:

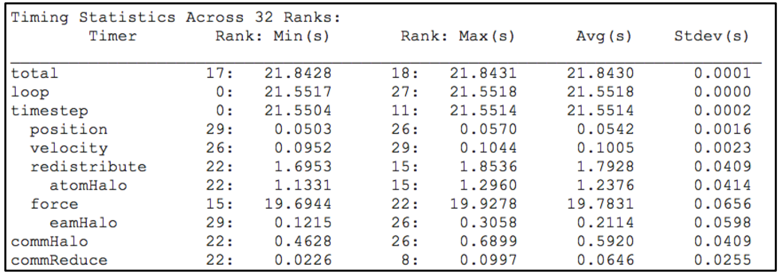
- 1Million Particles
- 100 Time-steps
- Mpi + OpenMP

V&V Information:

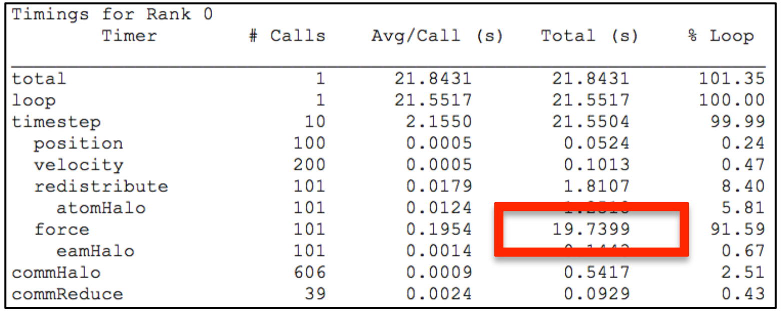


```
Simulation Validation:
  Initial energy  : -3.460523233094
  Final energy    : -3.460530068412
  eFinal/eInitial : 1.000002
  Final atom count : 1024000, no atoms lost
```

Output:



```
Timing Statistics Across 32 Ranks:
        Timer             Rank: Min(s)        Rank: Max(s)       Avg(s)     Stdev(s)

total                      17:   21.8428      18:   21.8431     21.8430      0.0001
loop                        0:   21.5517      27:   21.5518     21.5518      0.0000
timestep                    0:   21.5504      11:   21.5514     21.5514      0.0002
  position                 29:    0.0503      26:    0.0570      0.0542      0.0016
  velocity                 26:    0.0952      29:    0.1044      0.1005      0.0023
  redistribute             22:    1.6953      15:    1.8536      1.7928      0.0409
    atomHalo               22:    1.1331      15:    1.2960      1.2376      0.0414
  force                    15:   19.6944      22:   19.9278     19.7831      0.0656
    eamHalo                29:    0.1215      26:    0.3058      0.2114      0.0598
commHalo                   22:    0.4628      26:    0.6899      0.5920      0.0409
commReduce                 22:    0.0226       8:    0.0997      0.0646      0.0255
```



```
Timings for Rank 0
        Timer              # Calls    Avg/Call (s)   Total (s)    % Loop

total                           1       21.8431       21.8431     101.35
loop                            1       21.5517       21.5517     100.00
timestep                       10        2.1550       21.5504      99.99
  position                    100        0.0005        0.0524       0.24
  velocity                    200        0.0005        0.1013       0.47
  redistribute                101        0.0179        1.8107       8.40
    atomHalo                  101        0.0124        1.2516       5.81
  force                       101        0.1954       19.7399      91.59
    eamHalo                   101        0.0014        0.1443       0.67
commHalo                      606        0.0009        0.5417       2.51
commReduce                     39        0.0024        0.0929       0.43
```

EAM Force:

Summary of Todo List I:

1. Align Data & Hint Compiler:
    - pot->phi
    - pot->rho
    - s->atoms->U
    - pot->dfEmbed
    - pot->rhobar
    - “s->atoms->r”
    - “s->atoms->f” …done!

2. Convert So(AoA) to multiple SoA:
    - “s->atoms->r”
    - “s->atoms->f” …done!
3. Convert reduction in inner loop on:
    - “s->atoms->r” to “simd reduction”
    - “s->atoms->f” to “simd reduction”
    - Hint:
        - Introduce summation variable
            - “sum_R”
            - “sum_F”

EAM Force:

Summary of Todo II:

1. Convert “interpolate” subroutine to:

- SIMD-Enabled function
  - Part 1: Declaration
  - Part 2: Definition

Or:

- “omp” SIMD-Enabled function
  - Part 1: Declaration
  - Part 2: Definition
    - “omp declare simd”
    - “align clause”
    - “inbranch clause”

EAM Force:

<u>Summary of Todo III:</u>

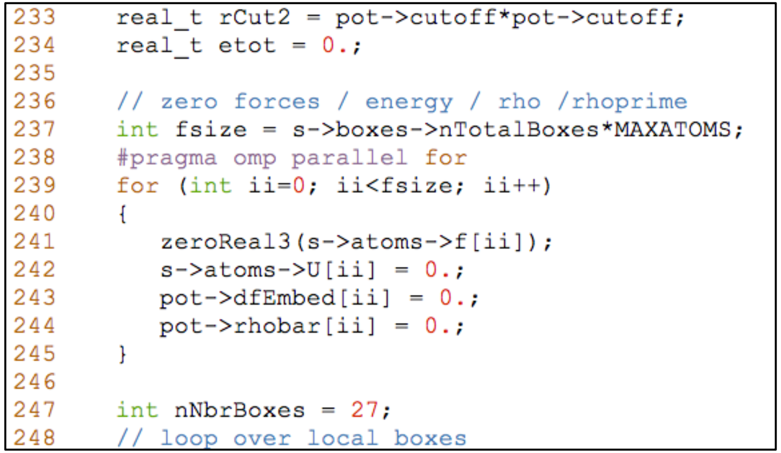
Hint compiler on alignment “eam.c”:

1. “table->values” variable:
    - @Function
        - InterpolationObject* initInterpolationObject

1. “buf” variable:
    - @Function
        - void eamReadSetfl
        - void eamReadFuncfl

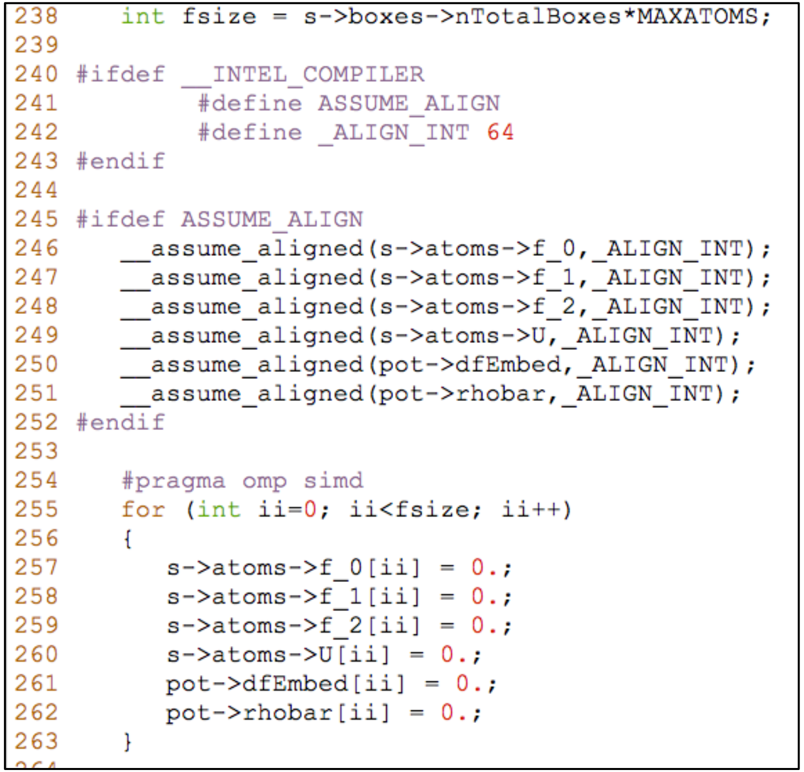
EAM Force:

Source Code Before:



```
    real_t rCut2 = pot->cutoff*pot->cutoff;
    real_t etot = 0.;

    // zero forces / energy / rho /rhoprime
    int fsize = s->boxes->nTotalBoxes*MAXATOMS;
    #pragma omp parallel for
    for (int ii=0; ii<fsize; ii++)
    {
       zeroReal3(s->atoms->f[ii]);
       s->atoms->U[ii] = 0.;
       pot->dfEmbed[ii] = 0.;
       pot->rhobar[ii] = 0.;
    }

    int nNbrBoxes = 27;
    // loop over local boxes
```

Source Code After:



```
   int fsize = s->boxes->nTotalBoxes*MAXATOMS;

#ifdef __INTEL_COMPILER
        #define ASSUME_ALIGN
        #define _ALIGN_INT 64
#endif

#ifdef ASSUME_ALIGN
   __assume_aligned(s->atoms->f_0,_ALIGN_INT);
   __assume_aligned(s->atoms->f_1,_ALIGN_INT);
   __assume_aligned(s->atoms->f_2,_ALIGN_INT);
   __assume_aligned(s->atoms->U,_ALIGN_INT);
   __assume_aligned(pot->dfEmbed,_ALIGN_INT);
   __assume_aligned(pot->rhobar,_ALIGN_INT);
#endif

   #pragma omp simd
   for (int ii=0; ii<fsize; ii++)
   {
      s->atoms->f_0[ii] = 0.;
      s->atoms->f_1[ii] = 0.;
      s->atoms->f_2[ii] = 0.;
      s->atoms->U[ii] = 0.;
      pot->dfEmbed[ii] = 0.;
      pot->rhobar[ii] = 0.;
   }
```

Modifications – Initialization step:

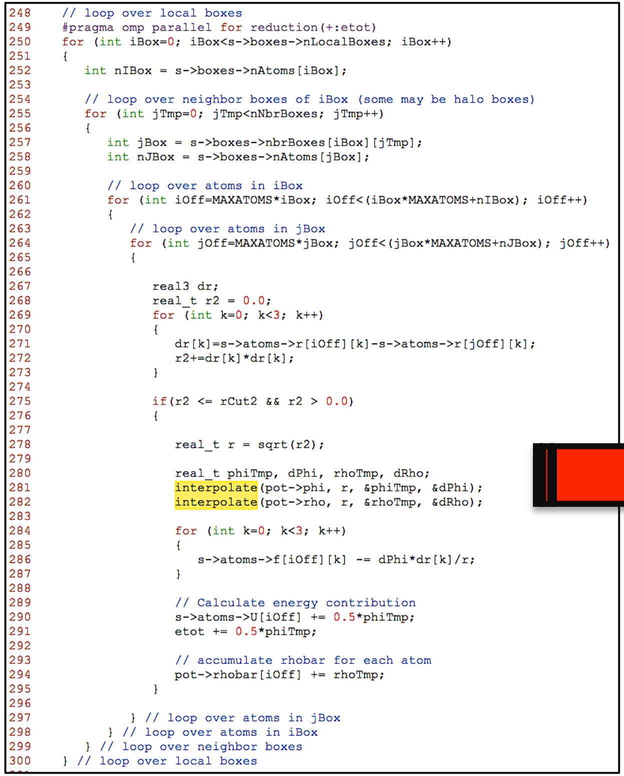
- Aligned memory allocation
  - Part 1: "__mm_malloc"
  - Part 2: "hint compiler"
- 1X(SoA(oA)) => 3X(SoA)

EAM Force

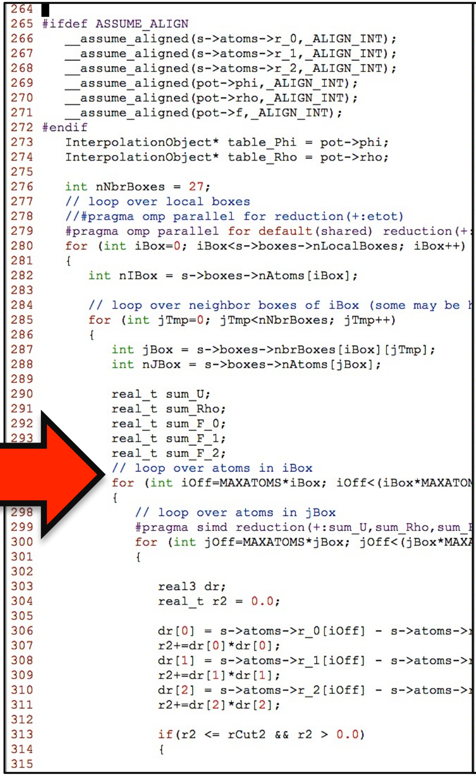
Source Code Before:

Modifications – Force Calculation Pass I:

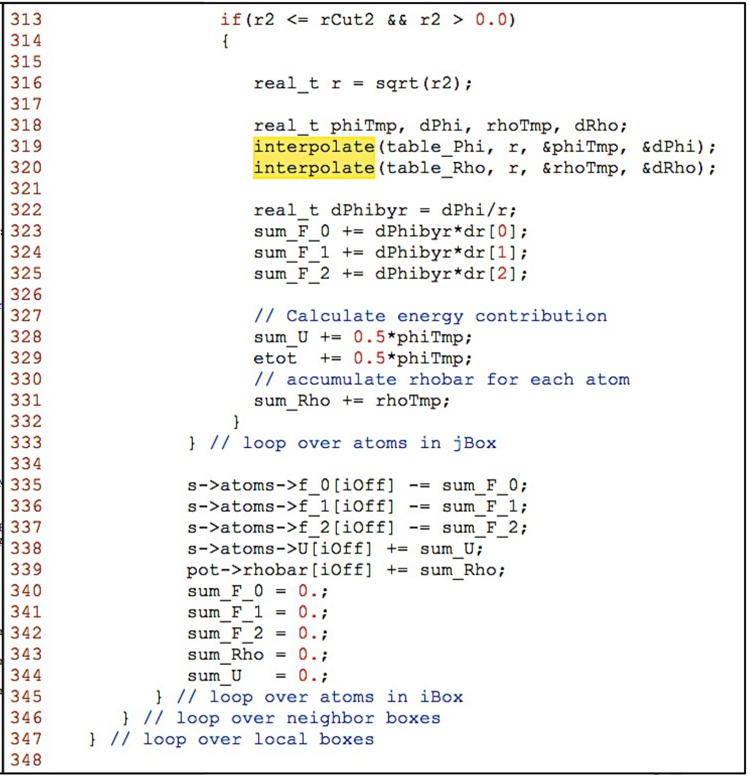
- Simd reduction
  - s->atom->U
  - s->atom->F_(all)
  - pot->rhobar



```c
   // loop over local boxes
   #pragma omp parallel for reduction(+:etot)
   for (int iBox=0; iBox<s->boxes->nLocalBoxes; iBox++)
   {
      int nIBox = s->boxes->nAtoms[iBox];

      // loop over neighbor boxes of iBox (some may be halo boxes)
      for (int jTmp=0; jTmp<nNbrBoxes; jTmp++)
      {
         int jBox = s->boxes->nbrBoxes[iBox][jTmp];
         int nJBox = s->boxes->nAtoms[jBox];

         // loop over atoms in iBox
         for (int iOff=MAXATOMS*iBox; iOff<(iBox*MAXATOMS+nIBox); iOff++)
         {
            // loop over atoms in jBox
            for (int jOff=MAXATOMS*jBox; jOff<(jBox*MAXATOMS+nJBox); jOff++)
            {

               real3 dr;
               real_t r2 = 0.0;
               for (int k=0; k<3; k++)
               {
                  dr[k]=s->atoms->r[iOff][k]-s->atoms->r[jOff][k];
                  r2+=dr[k]*dr[k];
               }

               if(r2 <= rCut2 && r2 > 0.0)
               {

                  real_t r = sqrt(r2);

                  real_t phiTmp, dPhi, rhoTmp, dRho;
                  interpolate(pot->phi, r, &phiTmp, &dPhi);
                  interpolate(pot->rho, r, &rhoTmp, &dRho);

                  for (int k=0; k<3; k++)
                  {
                     s->atoms->f[iOff][k] -= dPhi*dr[k]/r;
                  }

                  // Calculate energy contribution
                  s->atoms->U[iOff] += 0.5*phiTmp;
                  etot += 0.5*phiTmp;

                  // accumulate rhobar for each atom
                  pot->rhobar[iOff] += rhoTmp;
               }

            } // loop over atoms in jBox
         } // loop over atoms in iBox
      } // loop over neighbor boxes
   } // loop over local boxes
```



```c

#ifdef ASSUME_ALIGN
   __assume_aligned(s->atoms->r_0,_ALIGN_INT);
   __assume_aligned(s->atoms->r_1,_ALIGN_INT);
   __assume_aligned(s->atoms->r_2,_ALIGN_INT);
   __assume_aligned(pot->phi,_ALIGN_INT);
   __assume_aligned(pot->rho,_ALIGN_INT);
   __assume_aligned(pot->f,_ALIGN_INT);
#endif
   InterpolationObject* table_Phi = pot->phi;
   InterpolationObject* table_Rho = pot->rho;

   int nNbrBoxes = 27;
   // loop over local boxes
   //#pragma omp parallel for reduction(+:etot)
   #pragma omp parallel for default(shared) reduction(+:
   for (int iBox=0; iBox<s->boxes->nLocalBoxes; iBox++)
   {
      int nIBox = s->boxes->nAtoms[iBox];

      // loop over neighbor boxes of iBox (some may be h
      for (int jTmp=0; jTmp<nNbrBoxes; jTmp++)
      {
         int jBox = s->boxes->nbrBoxes[iBox][jTmp];
         int nJBox = s->boxes->nAtoms[jBox];

         real_t sum_U;
         real_t sum_Rho;
         real_t sum_F_0;
         real_t sum_F_1;
         real_t sum_F_2;
         // loop over atoms in iBox
         for (int iOff=MAXATOMS*iBox; iOff<(iBox*MAXATOM
         {
            // loop over atoms in jBox
            #pragma simd reduction(+:sum_U,sum_Rho,sum_F
            for (int jOff=MAXATOMS*jBox; jOff<(jBox*MAXA
            {

               real3 dr;
               real_t r2 = 0.0;

               dr[0] = s->atoms->r_0[iOff] - s->atoms->r
               r2+=dr[0]*dr[0];
               dr[1] = s->atoms->r_1[iOff] - s->atoms->r
               r2+=dr[1]*dr[1];
               dr[2] = s->atoms->r_2[iOff] - s->atoms->r
               r2+=dr[2]*dr[2];

               if(r2 <= rCut2 && r2 > 0.0)
               {

```



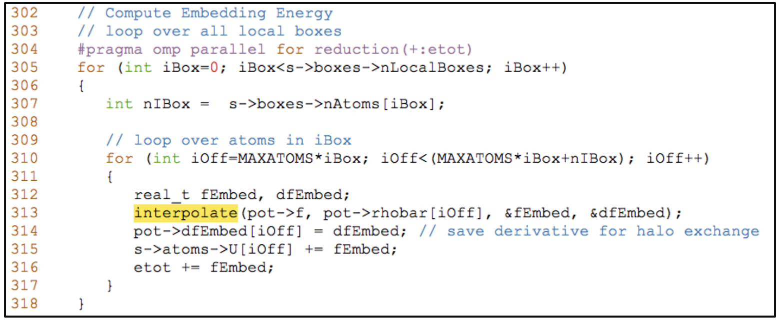
```c
               if(r2 <= rCut2 && r2 > 0.0)
               {

                  real_t r = sqrt(r2);

                  real_t phiTmp, dPhi, rhoTmp, dRho;
                  interpolate(table_Phi, r, &phiTmp, &dPhi);
                  interpolate(table_Rho, r, &rhoTmp, &dRho);

                  real_t dPhibyr = dPhi/r;
                  sum_F_0 += dPhibyr*dr[0];
                  sum_F_1 += dPhibyr*dr[1];
                  sum_F_2 += dPhibyr*dr[2];

                  // Calculate energy contribution
                  sum_U += 0.5*phiTmp;
                  etot  += 0.5*phiTmp;
                  // accumulate rhobar for each atom
                  sum_Rho += rhoTmp;
               }
            } // loop over atoms in jBox

            s->atoms->f_0[iOff] -= sum_F_0;
            s->atoms->f_1[iOff] -= sum_F_1;
            s->atoms->f_2[iOff] -= sum_F_2;
            s->atoms->U[iOff] += sum_U;
            pot->rhobar[iOff] += sum_Rho;
            sum_F_0 = 0.;
            sum_F_1 = 0.;
            sum_F_2 = 0.;
            sum_Rho  = 0.;
            sum_U    = 0.;
         } // loop over atoms in iBox
      } // loop over neighbor boxes
   } // loop over local boxes
```

EAM Force:

Modifications – Force Calculation Pass II:

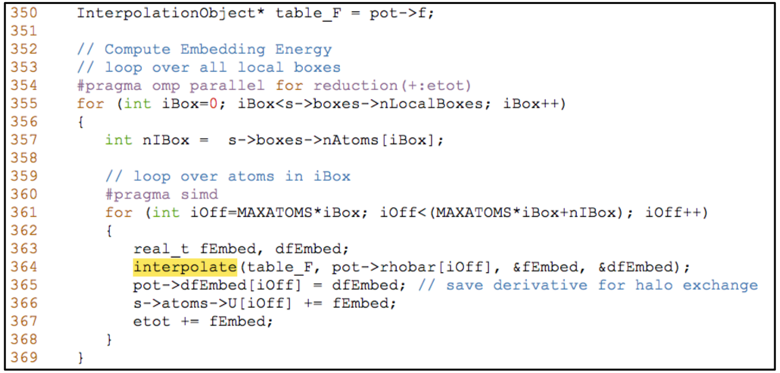
- Added "pragma simd" directive
  - s->atom->U

Source Code Before:



```
// Compute Embedding Energy
// loop over all local boxes
#pragma omp parallel for reduction(+:etot)
for (int iBox=0; iBox<s->boxes->nLocalBoxes; iBox++)
{
   int nIBox =  s->boxes->nAtoms[iBox];

   // loop over atoms in iBox
   for (int iOff=MAXATOMS*iBox; iOff<(MAXATOMS*iBox+nIBox); iOff++)
   {
      real_t fEmbed, dfEmbed;
      interpolate(pot->f, pot->rhobar[iOff], &fEmbed, &dfEmbed);
      pot->dfEmbed[iOff] = dfEmbed; // save derivative for halo exchange
      s->atoms->U[iOff] += fEmbed;
      etot += fEmbed;
   }
}
```

Source Code After:



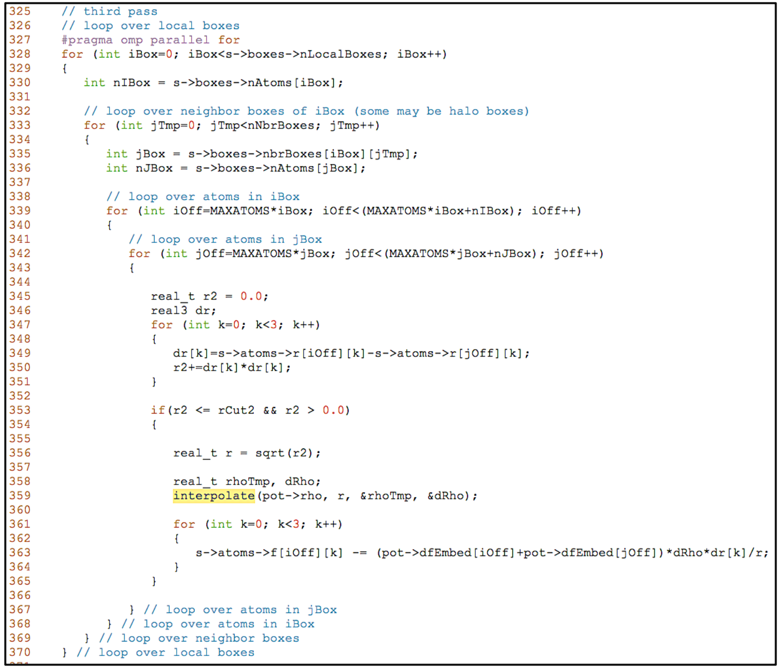
```
InterpolationObject* table_F = pot->f;

// Compute Embedding Energy
// loop over all local boxes
#pragma omp parallel for reduction(+:etot)
for (int iBox=0; iBox<s->boxes->nLocalBoxes; iBox++)
{
   int nIBox =  s->boxes->nAtoms[iBox];

   // loop over atoms in iBox
   #pragma simd
   for (int iOff=MAXATOMS*iBox; iOff<(MAXATOMS*iBox+nIBox); iOff++)
   {
      real_t fEmbed, dfEmbed;
      interpolate(table_F, pot->rhobar[iOff], &fEmbed, &dfEmbed);
      pot->dfEmbed[iOff] = dfEmbed; // save derivative for halo exchange
      s->atoms->U[iOff] += fEmbed;
      etot += fEmbed;
   }
}
```

EAM Force:

Modifications – Force Calculation Pass III:

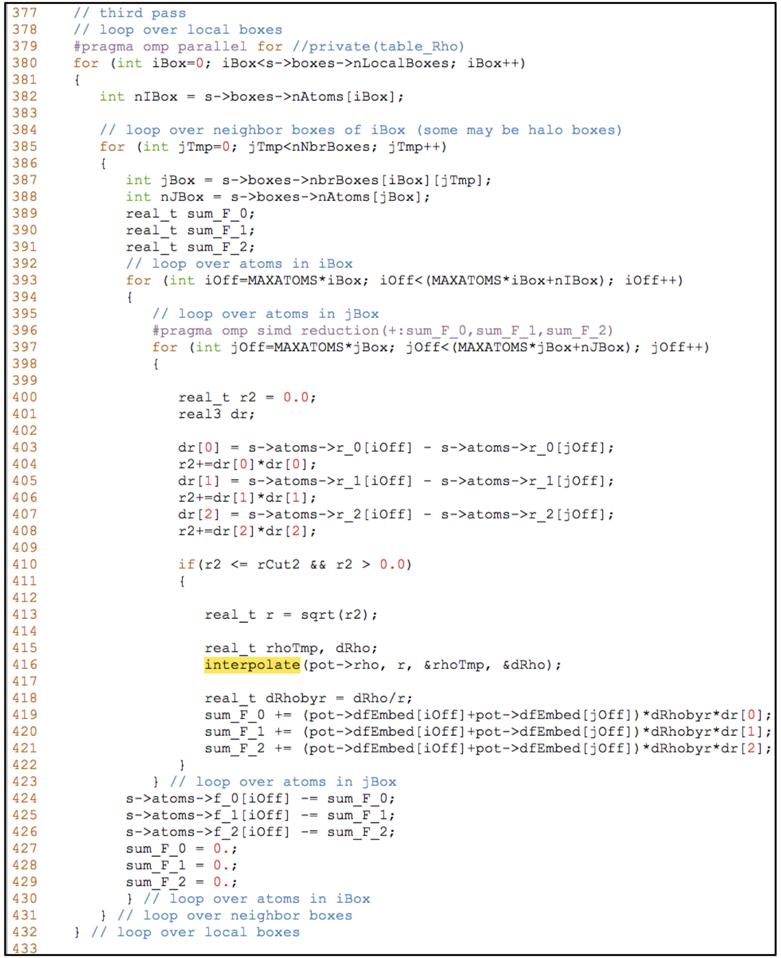
- Simd reduction: “s->atoms->f(All)”

Source Code Before:



```
// third pass
// loop over local boxes
#pragma omp parallel for
for (int iBox=0; iBox<s->boxes->nLocalBoxes; iBox++)
{
   int nIBox = s->boxes->nAtoms[iBox];

   // loop over neighbor boxes of iBox (some may be halo boxes)
   for (int jTmp=0; jTmp<nNbrBoxes; jTmp++)
   {
      int jBox = s->boxes->nbrBoxes[iBox][jTmp];
      int nJBox = s->boxes->nAtoms[jBox];

      // loop over atoms in iBox
      for (int iOff=MAXATOMS*iBox; iOff<(MAXATOMS*iBox+nIBox); iOff++)
      {
         // loop over atoms in jBox
         for (int jOff=MAXATOMS*jBox; jOff<(MAXATOMS*jBox+nJBox); jOff++)
         {

            real_t r2 = 0.0;
            real3 dr;
            for (int k=0; k<3; k++)
            {
               dr[k]=s->atoms->r[iOff][k]-s->atoms->r[jOff][k];
               r2+=dr[k]*dr[k];
            }

            if(r2 <= rCut2 && r2 > 0.0)
            {

               real_t r = sqrt(r2);

               real_t rhoTmp, dRho;
               interpolate(pot->rho, r, &rhoTmp, &dRho);

               for (int k=0; k<3; k++)
               {
                  s->atoms->f[iOff][k] -= (pot->dfEmbed[iOff]+pot->dfEmbed[jOff])*dRho*dr[k]/r;
               }
            }

         } // loop over atoms in jBox
      } // loop over atoms in iBox
   } // loop over neighbor boxes
} // loop over local boxes
```

Source Code After:



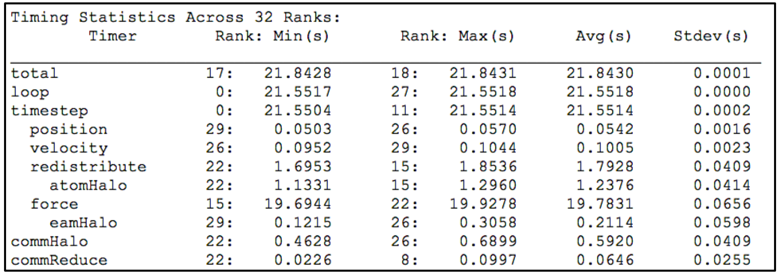
```
// third pass
// loop over local boxes
#pragma omp parallel for //private(table_Rho)
for (int iBox=0; iBox<s->boxes->nLocalBoxes; iBox++)
{
   int nIBox = s->boxes->nAtoms[iBox];

   // loop over neighbor boxes of iBox (some may be halo boxes)
   for (int jTmp=0; jTmp<nNbrBoxes; jTmp++)
   {
      int jBox = s->boxes->nbrBoxes[iBox][jTmp];
      int nJBox = s->boxes->nAtoms[jBox];
      real_t sum_F_0;
      real_t sum_F_1;
      real_t sum_F_2;
      // loop over atoms in iBox
      for (int iOff=MAXATOMS*iBox; iOff<(MAXATOMS*iBox+nIBox); iOff++)
      {
         // loop over atoms in jBox
         #pragma omp simd reduction(+:sum_F_0,sum_F_1,sum_F_2)
         for (int jOff=MAXATOMS*jBox; jOff<(MAXATOMS*jBox+nJBox); jOff++)
         {

            real_t r2 = 0.0;
            real3 dr;

            dr[0] = s->atoms->r_0[iOff] - s->atoms->r_0[jOff];
            r2+=dr[0]*dr[0];
            dr[1] = s->atoms->r_1[iOff] - s->atoms->r_1[jOff];
            r2+=dr[1]*dr[1];
            dr[2] = s->atoms->r_2[iOff] - s->atoms->r_2[jOff];
            r2+=dr[2]*dr[2];

            if(r2 <= rCut2 && r2 > 0.0)
            {

               real_t r = sqrt(r2);

               real_t rhoTmp, dRho;
               interpolate(pot->rho, r, &rhoTmp, &dRho);

               real_t dRhobyr = dRho/r;
               sum_F_0 += (pot->dfEmbed[iOff]+pot->dfEmbed[jOff])*dRhobyr*dr[0];
               sum_F_1 += (pot->dfEmbed[iOff]+pot->dfEmbed[jOff])*dRhobyr*dr[1];
               sum_F_2 += (pot->dfEmbed[iOff]+pot->dfEmbed[jOff])*dRhobyr*dr[2];
            }
         } // loop over atoms in jBox
      s->atoms->f_0[iOff] -= sum_F_0;
      s->atoms->f_1[iOff] -= sum_F_1;
      s->atoms->f_2[iOff] -= sum_F_2;
      sum_F_0 = 0.;
      sum_F_1 = 0.;
      sum_F_2 = 0.;
      } // loop over atoms in iBox
   } // loop over neighbor boxes
} // loop over local boxes
```

EAM Force – Optimized:

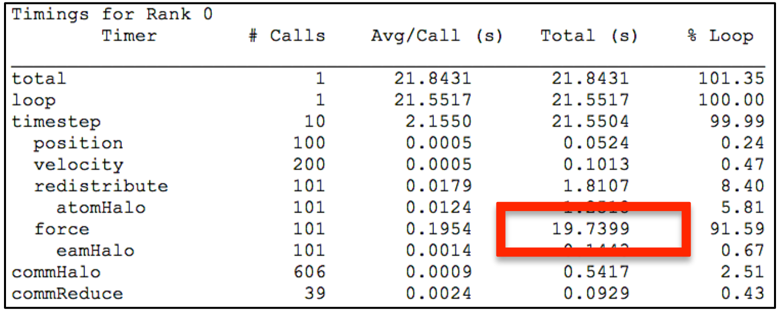
Performance Report”:

- ~27% Speedup in EAM Force kernel
- ~28% Speedup overall

Old Output:

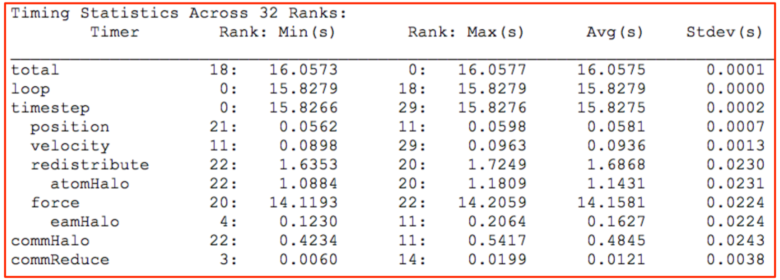


```
Timing Statistics Across 32 Ranks:
            Timer        Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)

total                    17:   21.8428      18:   21.8431    21.8430      0.0001
loop                      0:   21.5517      27:   21.5518    21.5518      0.0000
timestep                  0:   21.5504      11:   21.5514    21.5514      0.0002
  position               29:    0.0503      26:    0.0570     0.0542      0.0016
  velocity               26:    0.0952      29:    0.1044     0.1005      0.0023
  redistribute           22:    1.6953      15:    1.8536     1.7928      0.0409
    atomHalo             22:    1.1331      15:    1.2960     1.2376      0.0414
  force                  15:   19.6944      22:   19.9278    19.7831      0.0656
    eamHalo              29:    0.1215      26:    0.3058     0.2114      0.0598
commHalo                 22:    0.4628      26:    0.6899     0.5920      0.0409
commReduce               22:    0.0226       8:    0.0997     0.0646      0.0255
```

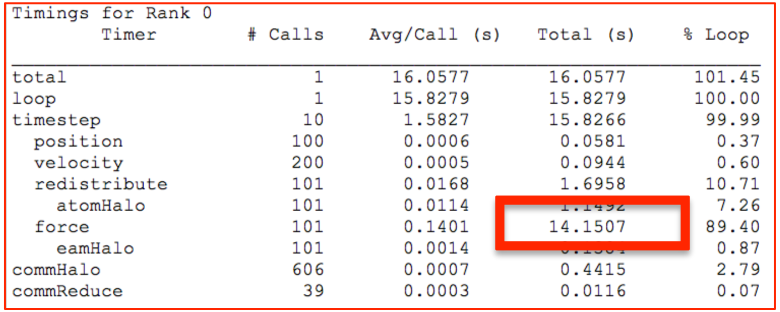


```
Timings for Rank 0
        Timer              # Calls    Avg/Call (s)   Total (s)    % Loop

total                            1       21.8431       21.8431    101.35
loop                             1       21.5517       21.5517    100.00
timestep                        10        2.1550       21.5504     99.99
  position                     100        0.0005        0.0524      0.24
  velocity                     200        0.0005        0.1013      0.47
  redistribute                 101        0.0179        1.8107      8.40
    atomHalo                   101        0.0124    [illegible]      5.81
  force                        101        0.1954       19.7399     91.59
    eamHalo                    101        0.0014        0.1443      0.67
commHalo                       606        0.0009        0.5417      2.51
commReduce                      39        0.0024        0.0929      0.43
```

New Output:



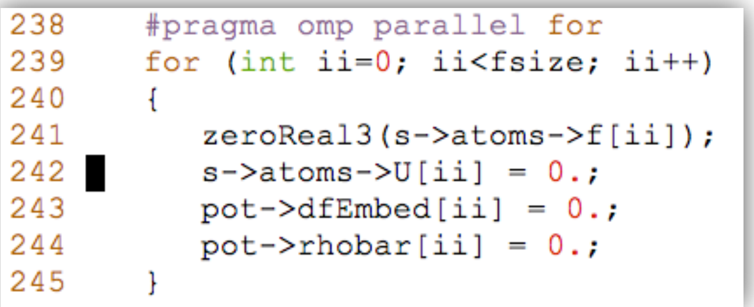
```
Timing Statistics Across 32 Ranks:
            Timer        Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)

total                    18:   16.0573       0:   16.0577    16.0575      0.0001
loop                      0:   15.8279      18:   15.8279    15.8279      0.0000
timestep                  0:   15.8266      29:   15.8276    15.8275      0.0002
  position               21:    0.0562      11:    0.0598     0.0581      0.0007
  velocity               11:    0.0898      29:    0.0963     0.0936      0.0013
  redistribute           22:    1.6353      20:    1.7249     1.6868      0.0230
    atomHalo             22:    1.0884      20:    1.1809     1.1431      0.0231
  force                  20:   14.1193      22:   14.2059    14.1581      0.0224
    eamHalo               4:    0.1230      11:    0.2064     0.1627      0.0224
commHalo                 22:    0.4234      11:    0.5417     0.4845      0.0243
commReduce                3:    0.0060      14:    0.0199     0.0121      0.0038
```



```
Timings for Rank 0
        Timer              # Calls    Avg/Call (s)   Total (s)    % Loop

total                            1       16.0577       16.0577    101.45
loop                             1       15.8279       15.8279    100.00
timestep                        10        1.5827       15.8266     99.99
  position                     100        0.0006        0.0581      0.37
  velocity                     200        0.0005        0.0944      0.60
  redistribute                 101        0.0168        1.6958     10.71
    atomHalo                   101        0.0114        1.1492      7.26
  force                        101        0.1401       14.1507     89.40
    eamHalo                    101        0.0014    [illegible]      0.87
commHalo                       606        0.0007        0.4415      2.79
commReduce                      39        0.0003        0.0116      0.07
```

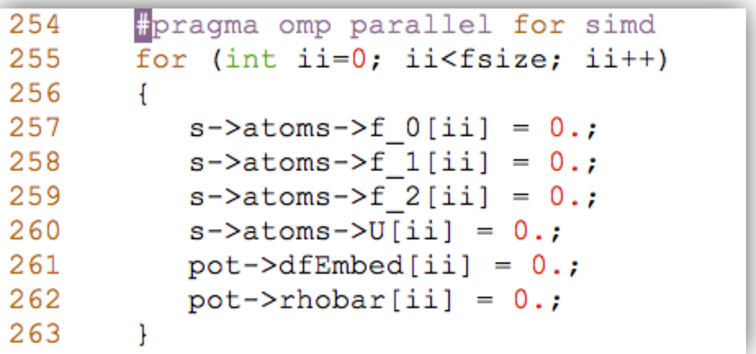
L-J Force – Revision:

Source Code Before:



```
#pragma omp parallel for
for (int ii=0; ii<fsize; ii++)
{
   zeroReal3(s->atoms->f[ii]);
   s->atoms->U[ii] = 0.;
   pot->dfEmbed[ii] = 0.;
   pot->rhobar[ii] = 0.;
}
```

Source Code After Revision II:



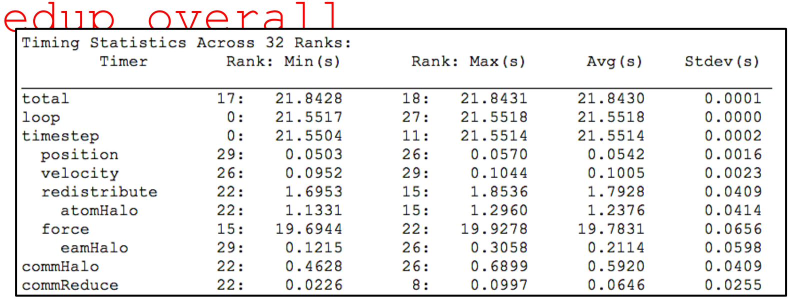
```
#pragma omp parallel for simd
for (int ii=0; ii<fsize; ii++)
{
   s->atoms->f_0[ii] = 0.;
   s->atoms->f_1[ii] = 0.;
   s->atoms->f_2[ii] = 0.;
   s->atoms->U[ii] = 0.;
   pot->dfEmbed[ii] = 0.;
   pot->rhobar[ii] = 0.;
}
```

EAM Force – Optimized

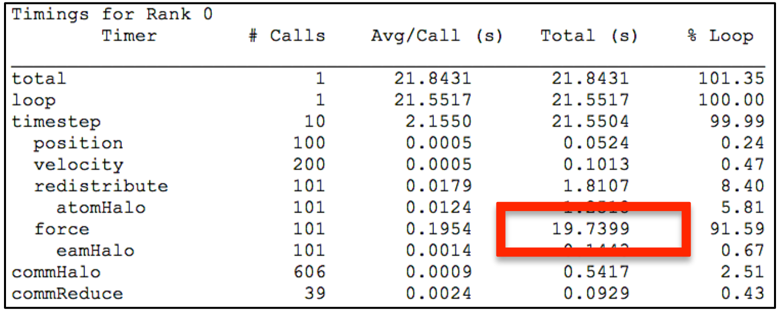
Revised Performance Report”:

- Revised Initialization Step
- ~40% Speedup in EAM Force Kernel
- ~32% Speedup overall

Old Output:

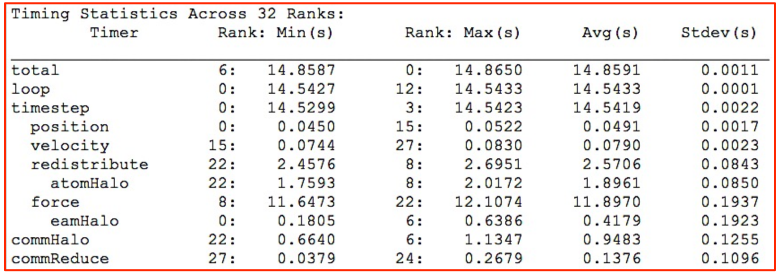


```
Timing Statistics Across 32 Ranks:
        Timer              Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total                      17:   21.8428       18:   21.8431    21.8430    0.0001
loop                        0:   21.5517       27:   21.5518    21.5518    0.0000
timestep                    0:   21.5504       11:   21.5514    21.5514    0.0002
  position                 29:    0.0503       26:    0.0570     0.0542    0.0016
  velocity                 26:    0.0952       29:    0.1044     0.1005    0.0023
  redistribute             22:    1.6953       15:    1.8536     1.7928    0.0409
    atomHalo               22:    1.1331       15:    1.2960     1.2376    0.0414
  force                    15:   19.6944       22:   19.9278    19.7831    0.0656
    eamHalo                29:    0.1215       26:    0.3058     0.2114    0.0598
commHalo                   22:    0.4628       26:    0.6899     0.5920    0.0409
commReduce                 22:    0.0226        8:    0.0997     0.0646    0.0255
```

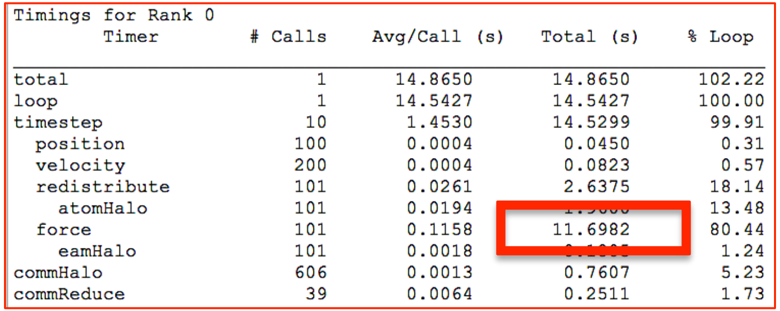


```
Timings for Rank 0
        Timer        # Calls    Avg/Call (s)   Total (s)    % Loop
___________________________________________________________________
total                      1     21.8431       21.8431     101.35
loop                       1     21.5517       21.5517     100.00
timestep                  10      2.1550       21.5504      99.99
  position               100      0.0005        0.0524       0.24
  velocity               200      0.0005        0.1013       0.47
  redistribute           101      0.0179        1.8107       8.40
    atomHalo             101      0.0124        1.2518       5.81
  force                  101      0.1954       19.7399      91.59
    eamHalo              101      0.0014        0.1443       0.67
commHalo                 606      0.0009        0.5417       2.51
commReduce                39      0.0024        0.0929       0.43
```

New Output:



```
Timing Statistics Across 32 Ranks:
        Timer              Rank: Min(s)       Rank: Max(s)      Avg(s)    Stdev(s)
_____________________________________________________________________________
total                       6:   14.8587        0:   14.8650    14.8591    0.0011
loop                        0:   14.5427       12:   14.5433    14.5433    0.0001
timestep                    0:   14.5299        3:   14.5423    14.5419    0.0022
  position                  0:    0.0450       15:    0.0522     0.0491    0.0017
  velocity                 15:    0.0744       27:    0.0830     0.0790    0.0023
  redistribute             22:    2.4576        8:    2.6951     2.5706    0.0843
    atomHalo               22:    1.7593        8:    2.0172     1.8961    0.0850
  force                     8:   11.6473       22:   12.1074    11.8970    0.1937
    eamHalo                 0:    0.1805        6:    0.6386     0.4179    0.1923
commHalo                   22:    0.6640        6:    1.1347     0.9483    0.1255
commReduce                 27:    0.0379       24:    0.2679     0.1376    0.1096
```



```
Timings for Rank 0
        Timer        # Calls    Avg/Call (s)   Total (s)    % Loop
___________________________________________________________________
total                      1     14.8650       14.8650     102.22
loop                       1     14.5427       14.5427     100.00
timestep                  10      1.4530       14.5299      99.91
  position               100      0.0004        0.0450       0.31
  velocity               200      0.0004        0.0823       0.57
  redistribute           101      0.0261        2.6375      18.14
    atomHalo             101      0.0194        1.9600      13.48
  force                  101      0.1158       11.6982      80.44
    eamHalo              101      0.0018        0.1805       1.24
commHalo                 606      0.0013        0.7607       5.23
commReduce                39      0.0064        0.2511       1.73
```

Questions & Comments
?/!