IXPUG
THE INTEL® XEON PHI™ USERS GROUP

Scaling and optimization results of the real-space DFT solver PARSEC on Haswell and KNL systems

Kevin Gott[1*], Charles Lena[2], Ariel Biller[3], Josh Neitzel[2], Kai-Hsin Liou[2], Jack Deslippe[1], James R Chelikowsky[2]

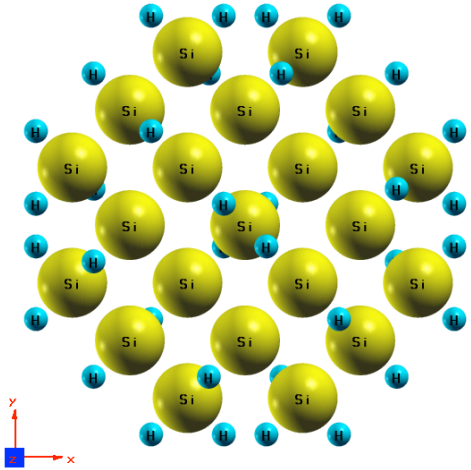
[1] NERSC, Lawrence Berkeley National Laboratory

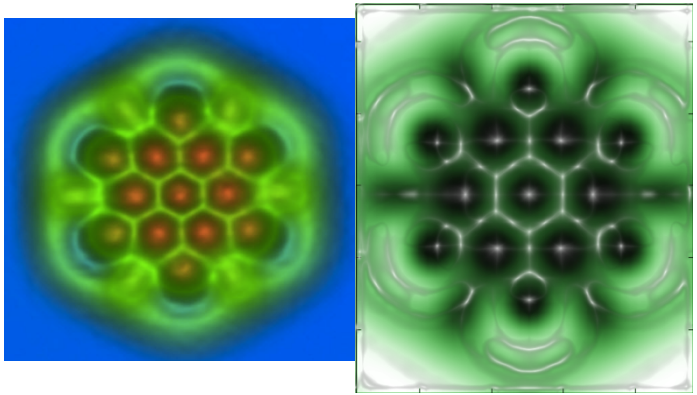
[2] University of Texas at Austin

[3] Weizmann Institute of Science

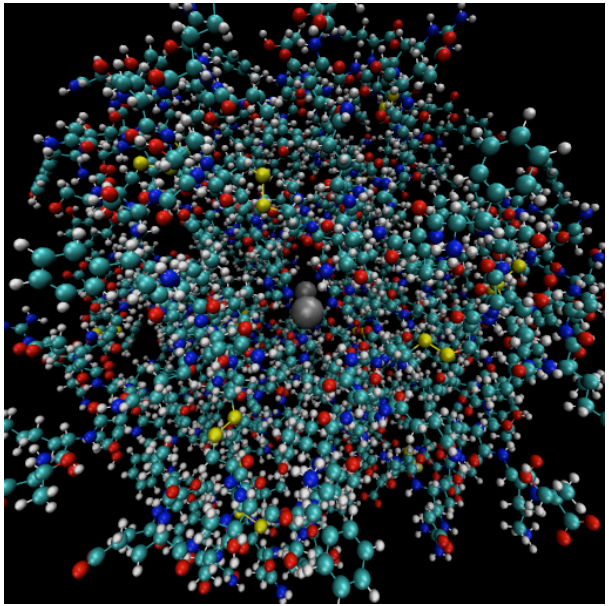
Motivations:
Quantum mechanics applied to materials

- Quantum mechanics can provide true electronic structure.
- Electronic structure often provides characteristics, properties & behavior.
- This method allows users to seek specific characteristics, properties, and behavior. Computational design of materials.





Non-contact AFM for hexabenzocoronene: (L) Measured (Science 9/12/2012), (R) Simulated using PARSEC, and colorized



Real space density functional theory (DFT) with pseudopotentials.

IXPUG
THE INTEL® XEON PHI™ USERS GROUP

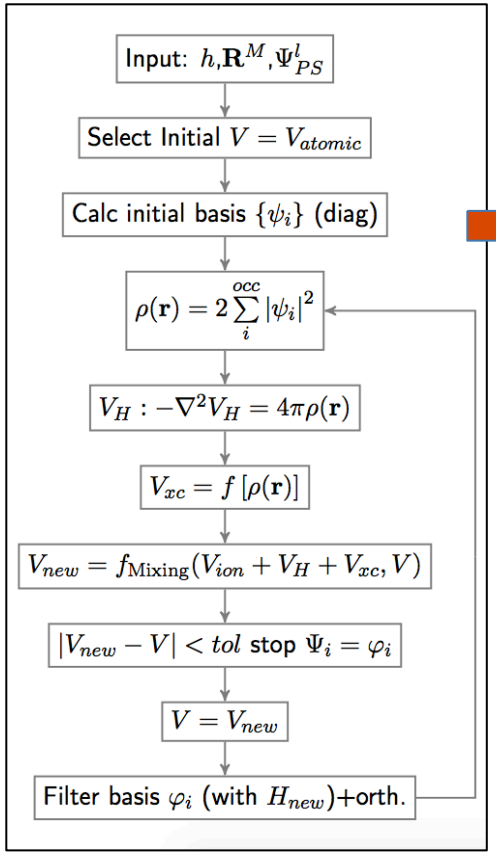
- Approximate exchange correlation w/ local density approximation
- Pseudopotentials -- only solve for valence electrons
- Simplify many body wave equation ==> $O(N_{elec}^3)$ ground state based only on an effective potential (the electronic density)
- Resulting nonlinear eigenproblem with Hamiltonian:

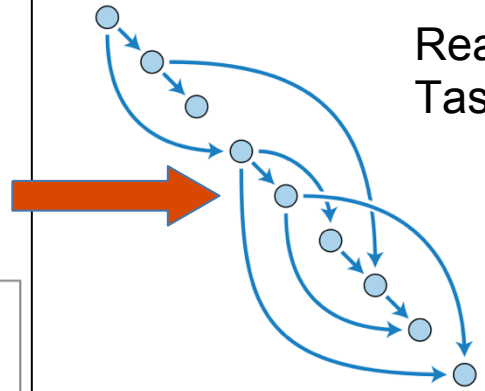
Governing Equations: Kohn-Sham Equations

$$\left[-\frac{1}{2}\nabla^2 + V_H\left[\rho(\mathbf{r})\right] + V_{ion}\left[\rho(\mathbf{r})\right] + V_{xc}\left[\rho(\mathbf{r})\right]\right]\psi_i(\mathbf{r}) = \lambda_i\psi_i(\mathbf{r})$$

$$\rho(\mathbf{r}) = 2\sum_{i}^{N_{occ}}\left|\psi_i(\mathbf{r})\right|^2$$

PARSEC – Pseudopotential Algorithms for Real Space Electronic Calculations



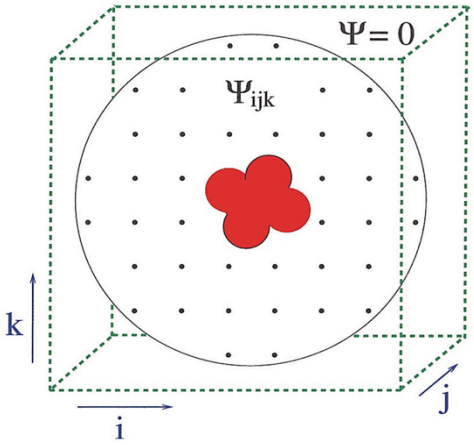


Real space DFT solver using Fortran
Task-based, fine-grained algorithm
Data/Task dependency = massive parallelism
(Better performance on exascale)

Uses a tasked vectorwise Hamiltonian

"Iterate, roughly in this order":

- Sync wavefunction data (for stencils)
- Perform sparse matrix-vector product
- Advance to next wavefunction



Subspace Iteration & Important Details

FLTR: degree *m* Chebyshev polynomial filtering of Psi through recursion

$$\mathbf{X}_{k+1} = \frac{2\sigma_{k+1}}{e}(\mathbf{H} - c\mathbf{I})\mathbf{X}_k - \sigma_{k+1}\sigma_k\mathbf{X}_{k-1}$$

ORTH: orthonormalization of Psi[:,1:s]

$$\psi_k^{\perp} = \psi_k - \sum_{j=1}^{k-1} \frac{\langle \psi_j^{\perp}, \psi_k \rangle}{\langle \psi_j^{\perp}, \psi_j^{\perp} \rangle} \psi_j^{\perp}$$

PROJ: projection of orthonormal basis

$$G_{ij} = \psi_i^* H \psi_j$$

DCMP: eigendecomposition of quotient matrix

$$GW = DW$$

UPDT: vectors to correct the orthonormal basis

$$\psi = \psi W$$

PROJ, DCMP, and UPDT form a Rayleigh-Ritz process

Overview of FLTR kernel workflow

Algorithm 2: Scaled Chebyshev filtering algorithm from PARSEC.

Input: $V \in \mathbb{R}^{N \times s}$, $H : \mathbb{R}^N \to \mathbb{R}^N$, Chebyshev polynomial order m, lowest bound a, lower bound a_L, upper bound b

Output: Filtered V

allocate *blk* comm. buffers // $a_L = a$ in nonscaled vers.
$n_{blk} = s/blk$; $\kappa(1:m) = n_{blk}/m$; $e = (b-a)/2$; $e_{rp} = 2/e$;
$c = (b+a)/2$; $\sigma = e/(c-a_L)$; $\sigma_1 = \sigma$;
$\sigma_{ei} = \sigma_1/e$; $\sigma_2 = 0$; vk = 1 - blk;
Aligned fastmem allocate $V_{\{1\}}, V_{\{2\}}, V_{\{3\}} \in \mathbb{C}^{N \times blk}$ **do** k_m = *1,m*
  do $\kappa_{block} = 1$: κ_{k_m}
    vk = vk + blk; iteratively prime *blk* buffers
    $V_{\{1\}} = V_{vk:vk+blk-1}$;
    $V_{\{2\}} = H(V_{\{1\}}, \text{buffers})$;
    $V_{\{2\}} = (V_{\{2\}} - c * V_{\{1\}}) * \sigma_{ei}$;
    do i = *2, pm(k_m)* // each degree is dictated by the pm array
      iteratively prime *blk* buffers;
      $V_{\{3\}} = H(V_{\{2\}}, \text{buffers})$; $\sigma_2 = 1/(2/\sigma_1 - \sigma)$;
      $V_{\{3\}} = (V_{\{3\}} - c * V_{\{2\}}) * e_{rp}$;
      $V_{\{3\}} = V_{\{3\}} - \sigma * V_{\{1\}}$;
      $V_{\{1\}} = V_{\{2\}}$;
      $V_{\{2\}} = V_{\{3\}} * \sigma_2$;
      $\sigma = \sigma_2$;
    $V_{vk:vk+blk-1} = V_{\{2\}}$
  sigma $= e/(c-a_L)$
3.1 cleanup buffers;

$$\mathbf{X}_{k+1} = \frac{2\sigma_{k+1}}{e}(\mathbf{H} - c\mathbf{I})\mathbf{X}_k - \sigma_{k+1}\sigma_k\mathbf{X}_{k-1}$$

- Hamiltonian **H** - sparse matrix-vector product
- **X** - block of subspace
- **H** was redesigned for OpenMP Task-based stencil operations, designed for massive parallelism for extremely large systems.

Overview of OpenMP tasked workflow


```
do k=1,blksize
!$OMP TASK FIRSTPRIVATE(.....)
   1st tier function call
     (or small conditional selection)
!$OMP FLUSH
!$OMP END TASK
enddo
```

```
!$OMP DECLARE SIMD(z222_DUDU) UNIFORM(cDiag,cU)
        do inner=1,8
           outvec(inner) = cDiag * invec(inner) + outvec(inner)
        enddo
       outvec(1:2)=invec(3:4)*cU+outvec(1:2)
       outvec(5:6)=invec(7:8)*cU+outvec(5:6)
```

```
select case (edgeval)
 case(1)
   call d222_DUL   (X2,X1,X3, invec,outvec)
 case(2)
   call d222_DUL   (X2,X3,X1, invec,outvec)
… (17 options, multiple stencil functions, some overlap)
 end select
```

(1) Tasked, block size call

(2) to a conditional function call statement
(edge/center, real/complex, etc…)

(3) which calculates using the desired stencil.

PARSEC vs. Compilers (& Performance Tools)

(1) ASSUME_ALIGNED gave substantial speedup in FLTR functions, (allowed AVX512 vectorization), despite optrpt stating the alignment statement was disregarded.

```
subroutine d222_DUL(cDiag,cU,cL,invec,outvec) !{{{
!$OMP DECLARE SIMD(d222_DUL) UNIFORM(cDiag,cU,cL)
PARAMETER DEFINITION
!DIR$ ASSUME_ALIGNED invec: 64
!DIR$ ASSUME_ALIGNED outvec: 64
      do inner=1,4
         outvec(inner+shft) = cDiag * invec(inner+shft) + outvec(inner+shft)
         outvec(inner)      =    cU * invec(inner+shft) + outvec(inner)
         outvec(inner)      = cDiag * invec(inner)      + outvec(inner)
         outvec(inner+shft) =    cL * invec(inner)      + outvec(inner+shft)
      enddo
end subroutine d222_DUL !}}}
```

(2) Misplaced ITAC global constants (COMMON block) library had a similar effect.

```
#ifdef ITAC
  include 'VT.inc'
  include 'vtcommon.inc'
#endif
 include 'vtcommon.inc'
```

PARSEC Hyper-threading Performance

FLTR Timings	Physical (32)		Intel HT (64)			
MPI RANKS	Average - SECONDS	StDev - SECONDS	Average - SECONDS	StDev - SECONDS	HyperThread Speedup	Runtime Reduction
1	22.22	4.61	15.26	1.53	1.46	31.33%
2	18.34	0.23	14.90	4.56	1.23	18.79%
4	20.10	3.97	11.81	0.42	1.70	41.27%
8	16.64	2.68	12.16	1.79	1.37	26.97%
16	19.08	3.40	14.45	2.13	1.32	24.27%
32	19.25	3.60	14.13	0.56	1.36	26.60%

Cori Haswell

Cori KNL

Average - SECONDS	TOTAL			Speedup	
MPI RANKS	68	136	272	2 HT	4 HT
68	24.81	19.46	16.58	1.28	1.50
34	24.48	17.43	14.91	1.40	1.64
17	21.28	14.53	16.36	1.46	1.30
16			65.30		
4			22.80		
2		23.52	28.96		
1			39.32		

PARSEC shows good hyper-threading performance on both Haswell and KNL.

Results from APS March Meeting 2017 simulating a Silicon nanocluster.

KNL and Work Throughput

When utilizing appropriate algorithms, the primary efficiency hurdle is supplying the appropriate amount of computing power for a given problem size or vice versa.

Si28H36	MPI	TOTAL HW THREADS			SPEEDUP	
Hamil. Rank	RANKS	68	136	272	2 HT	4 HT
855568	1	11.55	8.52	7.63	1.36	1.51
	2	15.33	15.1	17.45	1.02	0.88
	4	16.04	16.39	19.45	0.98	0.82
1575616	1	21.52	15.88	14.18	1.36	1.52
	2	27.83	25.64	28.165	1.09	0.99
	4	27.11	25.915	29.225	1.05	0.93

Motivations of the Force Update

$$F_a^\alpha = \int \rho(\mathbf{r}) \frac{\partial V_{\text{loc}}(r_a)}{\partial r_a^\alpha} d^3r + 2 \sum_{n,lm} \langle \Delta V_{lm}^a \rangle G_{n,lm}^a \frac{\partial G_{n,lm}^a}{\partial r_a^\alpha} - \frac{\partial E_{i\text{-}i}}{\partial R_a^\alpha}.$$

Local Forces | Non-Local Forces

- Forces are a **critical output parameter** and a **method of comparison** while testing.
- Previously **entirely unthreaded & poorly optimized** (other kernels took precedence).
- Want to implement in a way that **allows for future changes** (e.g. higher-order forces).
- Expected to take comparatively **negligible amount of time**, but:
- At large numbers of atoms or in complex systems, **becomes prohibitively expensive**.
- Can be substantially improved with **extreme parallelism practices on KNL**.

Original Non-Local Force Calculation

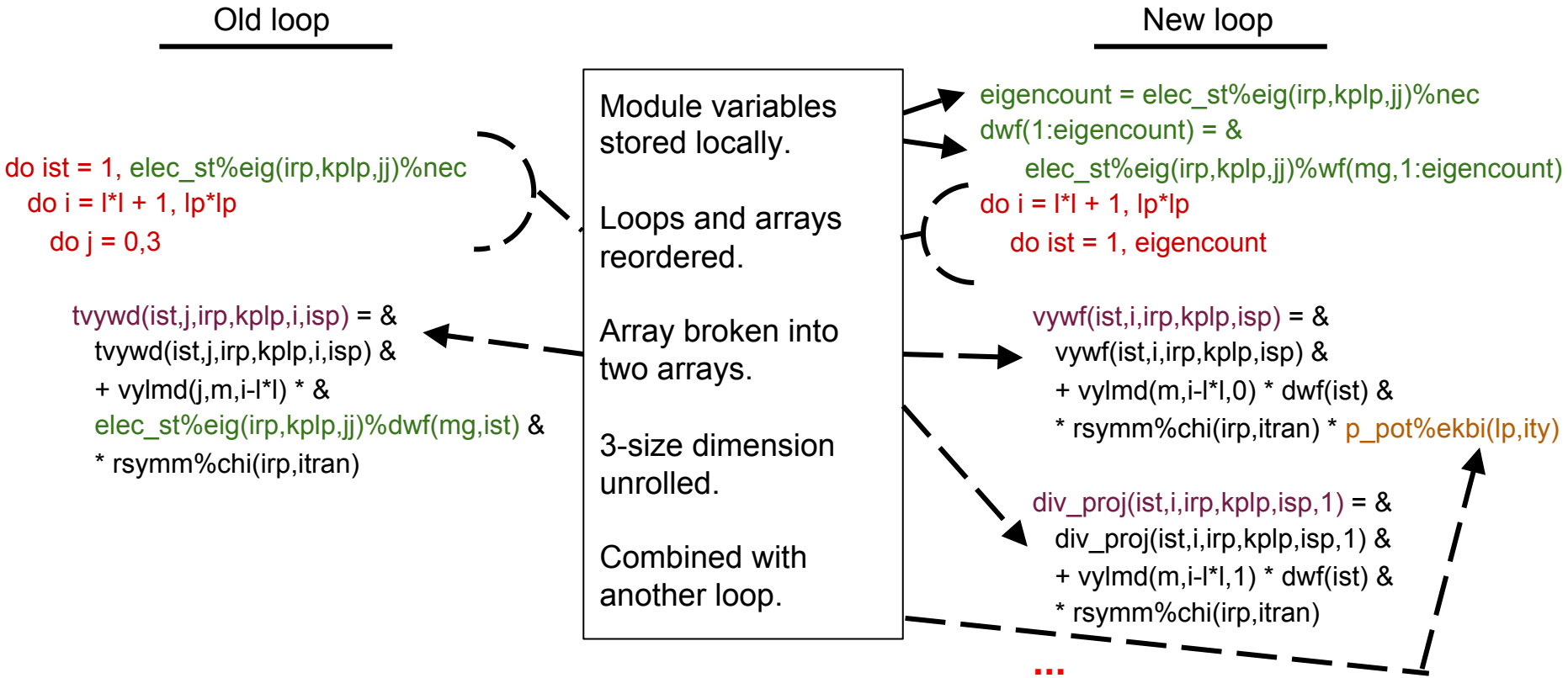
Original Force Pseudocode:

```
do type
   do atom
      calc A & B
      reduceAll A & B to master
      calc Δforce = f(A&B) on master
      store force on master
   end atom
end type
```

Important Features

- **Each atom done completely** due to size of A & B (30000 reals each at least, much higher with more complex features) vs. size of force (3 reals).
- **Math includes ∑A * ∑B**, so both sums are completed and sent to the master rank and remaining work is done there.
- **Synchronized work**: Each atom is done one at a time.
- **Sparse work**: Physical space is split along ranks, so for a typical distribution, most ranks do not have data to contribute to each atom.

Non-local Force Calculation Loop Changes



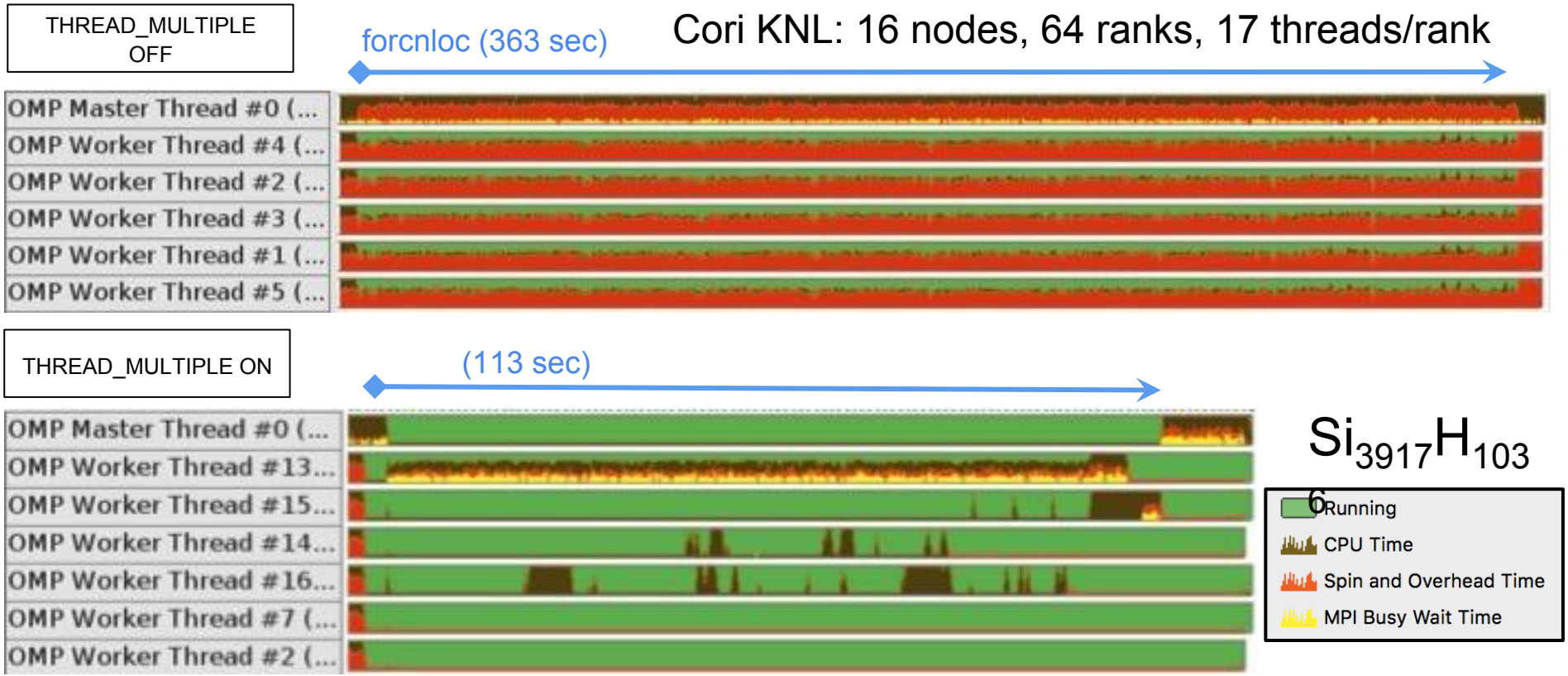
Improved Non-local Force Calculation

```
do type
   BUILD COMMS:
   MPI_COMM_SPLIT(atom, rank_has_data)
   !$OMP DO
   do atom
      if comm(atom) = MPI_COMM_NULL, cycle
      calc A & B
      reduceAll(comm(atom), A)
      calc Δforce = f(A&B)
      reduceAll(comm(atom), Δforce)
      store locally with master of comm(atom)
   end atom
   !$OMP END DO
end type
reduceAll(world, Δforce)
do atom
   calc force = force + Δforce on master
end atom
```

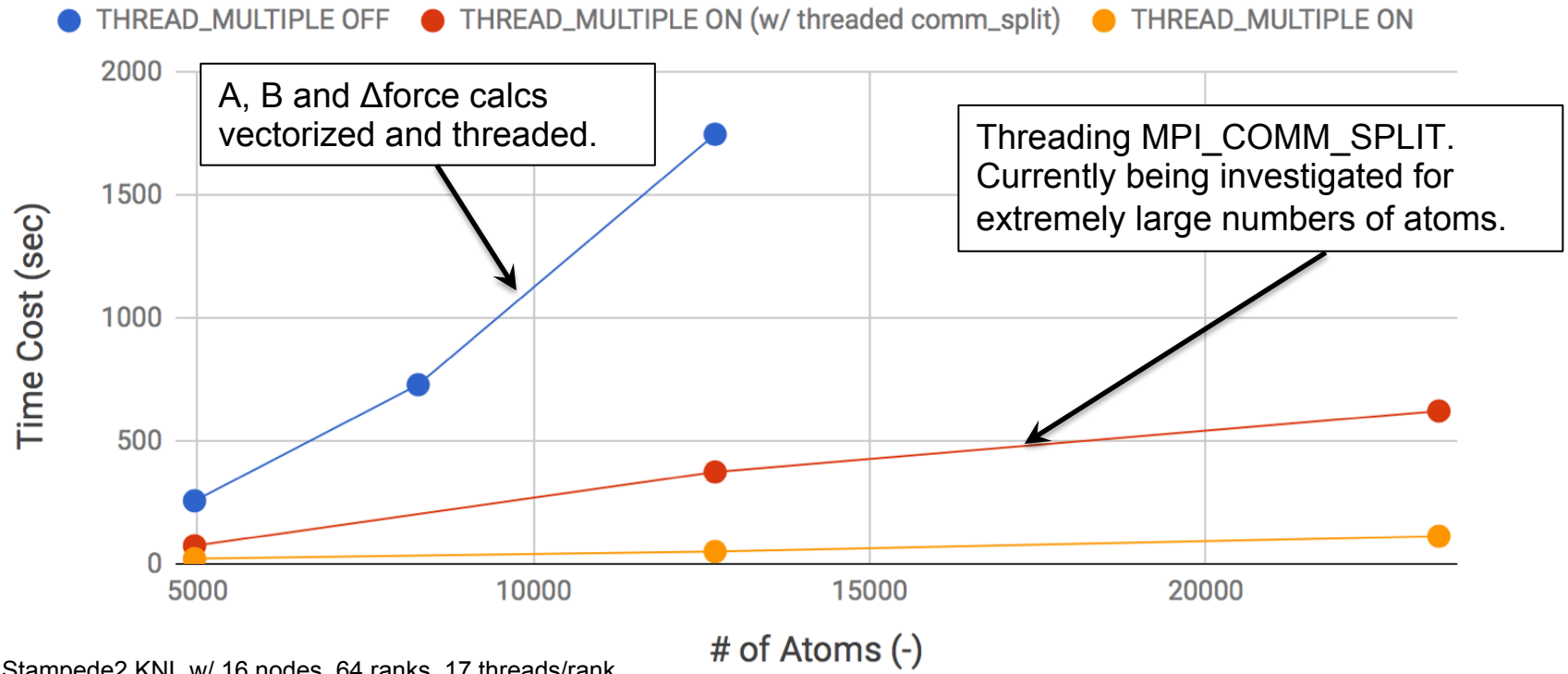
Key Changes

- **Implemented THREAD_MULTIPLE**.
- **Atom loop is threaded**, allowing multiple atoms to be solved simultaneously.
- Preemptively create an **array of comms**, one for each atom, that allow mpi ranks without data to move to the next atom.
- **Final storage** on the master is performed **outside the main calculation loop**.
- For larger problems (with # of atoms > max MPI comms): Added **an additional loop to create and free comms** using large atom batches.

Threading Comparison (VTune)

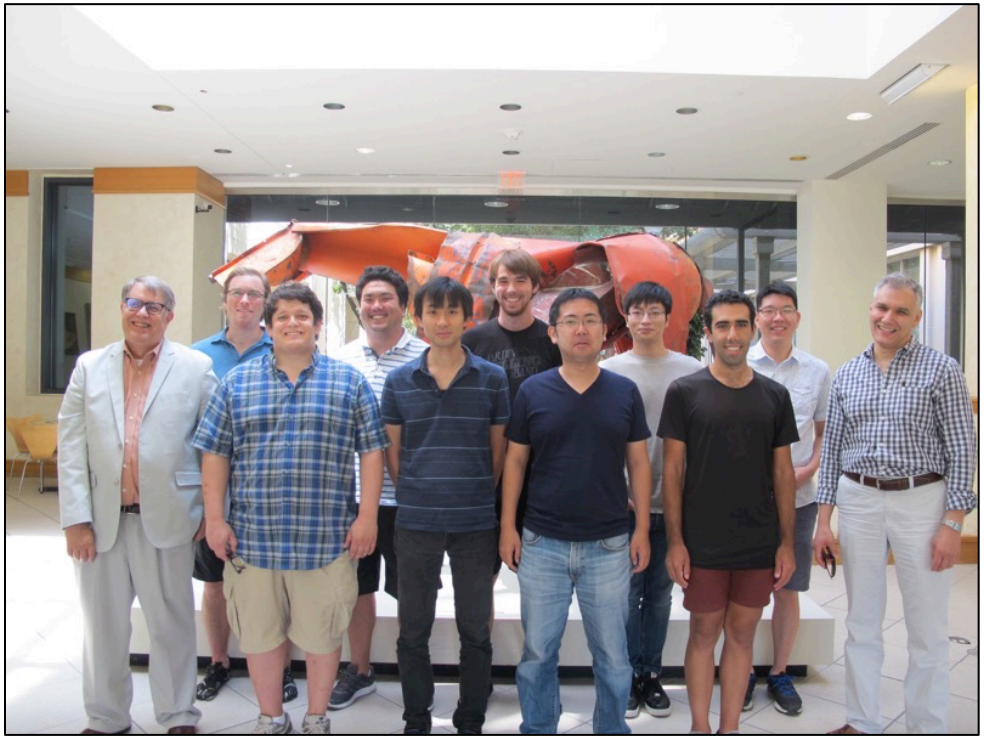


Non-Local Force Scaling Results



5k atoms: Stampede2 KNL w/ 16 nodes, 64 ranks, 17 threads/rank
8k atoms: Stampede2 KNL w/ 32 nodes, 128 ranks, 17 threads/rank
13k atoms: Stampede2 KNL w/ 64 nodes, 256 ranks, 17 threads/rank
23k atoms: Stampede2 KNL w/ 256 nodes, 512 ranks, 34 threads/rank

Acknowledgements



The CCM Team at University of Texas, Austin

NATIONAL SCIENCE FOUNDATION

IXPUG
THE INTEL® XEON PHI™ USERS GROUP

ICES

NERSC

THE UNIVERSITY OF TEXAS AT AUSTIN

CCM
CENTER FOR COMPUTATIONAL MATERIALS

THE Welch FOUNDATION
Advancing Chemistry. Improving Life.

DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA

TACC

Extra Slides

Threaded MPI_COMM_SPLIT Methodology

```
!$OMP PARALLEL DO DEFAULT(NONE)&
do thread = 1, omp_threads
  do atomsubset = 1, atomsperthread
    iat = calc subset index( thread, atomsubset )
    ja = calc global index( thread, atomgroup, atomsubset, type )
    if (has_data(ja)) then
      call MPI_COMM_SPLIT(commworld(thread), 1, 0, comm(iat), err)
      rank_has_data = 1
    else
      call MPI_COMM_SPLIT(commworld(thread), MPI_UNDEFINED,
                                                  0, comm(iat), err)
!$OMP END PARALLEL DO
```

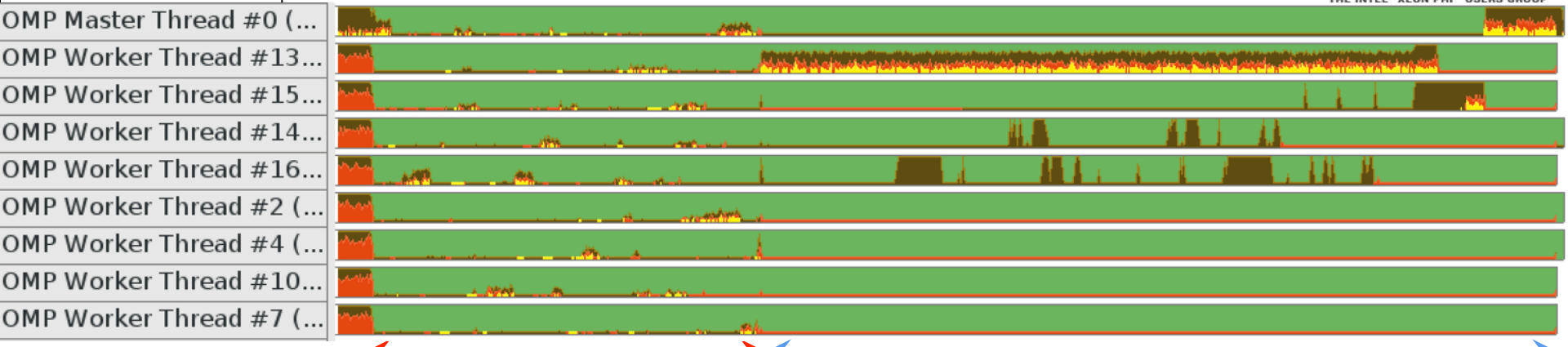
- Each **thread gets a duplicate** of **MPI_COMM_WORLD** to simultaneously communicate through.
- Requires a **thread loop & subset loop** to properly index the atom.
- **Key = 1: has data**, add your rank to this atom's comm.

  Key = MPI_UNDEFINED: has no data, build a MPI_NULL_COMM.
- **Currently, serial runs faster than threaded.**

THREAD_MULTIPLE ON
(w/ threaded comm_split)

$Si_{3917}H_{1036}$ Cori KNL: 16 nodes, 64 ranks, 17 threads/rank

IXPUG
THE INTEL® XEON PHI™ USERS GROUP



51 sec (comm_split)

106 sec

THREAD_MULTIPLE ON

113 sec