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 hexabenzocorone: (L) Measured (Science 9/12/2012), (R) Simulated using PARSEC, and colorized
Real space density functional theory (DFT) with pseudopotentials.

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

\[
\begin{align*}
\left[ -\frac{1}{2} \nabla^2 + V_H \left[ \rho(\mathbf{r}) \right] + V_{\text{ion}} \left[ \rho(\mathbf{r}) \right] + V_{xc} \left[ \rho(\mathbf{r}) \right] \right] \psi_i(\mathbf{r}) &= \lambda_i \psi_i(\mathbf{r}) \\
\rho(\mathbf{r}) &= 2 \sum_{i=1}^{N_{\text{occ}}} |\psi_i(\mathbf{r})|^2
\end{align*}
\]
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

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

**PROJ**: projection of orthonormal basis

**DCMP**: eigendecomposition of quotient matrix

**UPDT**: vectors to correct the orthonormal basis

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

$$X_{k+1} = \frac{2\sigma_{k+1}}{e} (H - cI)X_k - \sigma_{k+1}\sigma_k X_{k-1}$$

$$\psi_k = \psi_k - \sum_{j=1}^{k-1} \left( \frac{\langle \psi_j, \psi_k \rangle}{\langle \psi_j, \psi_j \rangle} \right) \psi_j$$

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

$$GW = DW$$

$$\psi = \psi W$$
Overview of FLTR kernel workflow

Algorithm 2: Scaled Chebyshev filtering algorithm from PARSEC.

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

Output: Filtered $V$

 allocate blk comm. buffers \;& // \; a_L = a \text{ in nonscaled vers.}
 $n_{blk} = s/blk; \; \kappa(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; \; v_k = 1 - blk;

Aligned fastmem allocate $V_{[1]}, V_{[2]}, V_{[3]} \in \mathbb{C}^{N \times blk}$ do $k_m = 1,m$

\[
X_{k+1} = \frac{2\sigma_{k+1}}{e} (H - cI)X_k - \sigma_{k+1} \sigma_k X_{k-1}
\]

- 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

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
```

```c
!$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)
```

(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.

```fortran
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.

```fortran
#ifndef ITAC
#include 'VT.inc'
#include 'vtcommon.inc'
#endif
```
PARSEC shows good hyper-threading performance on both Haswell and KNL.

Results from APS March Meeting 2017 simulating a Silicon nanocluster.
When utilizing appropriate algorithms, the primary efficiency hurdle is supplying the appropriate amount of computing power for a given problem size or vice versa.

<table>
<thead>
<tr>
<th>Si28H36 Hamil. Rank</th>
<th>MPI RANKS</th>
<th>TOTAL HW THREADS</th>
<th>SPEEDUP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>68</td>
<td>136</td>
</tr>
<tr>
<td>855568</td>
<td>1</td>
<td>11.55</td>
<td>8.52</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>15.33</td>
<td>15.1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>16.04</td>
<td>16.39</td>
</tr>
<tr>
<td>1575616</td>
<td>1</td>
<td>21.52</td>
<td>15.88</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>27.83</td>
<td>25.64</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>27.11</td>
<td>25.915</td>
</tr>
</tbody>
</table>
Motivations of the Force Update

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 $\sum A * \sum B$, 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

Old loop

do ist = 1, elec_st%eig(irp,kplp,jj)%nec
  do i = l*l + 1, lp*lp
    do j = 0,3
      tvywd(ist,j,irp,kplp,i,isp) = &
      tvywd(ist,j,irp,kplp,i,isp) &
      + vylmd(j,m,i-l*l) * &
      elec_st%eig(irp,kplp,jj)%dwf(mg,ist) &
      * rsymm%chi(irp,itran)

do j = 0,3

tvywd(ist,j,irp,kplp,i,isp) = &
  tvywd(ist,j,irp,kplp,i,isp) &
  + vylmd(j,m,i-l*l) * &
  elec_st%eig(irp,kplp,jj)%dwf(mg,ist) &
  * rsymm%chi(irp,itran)

New loop

eigencount = elec_st%eig(irp,kplp,jj)%nec
  dwf(1:eigencount) = &
  elec_st%eig(irp,kplp,jj)%wf(mg,1:eigencount)
  do i = l*l + 1, lp*lp
    do ist = 1, eigencount
      vywf(ist,i,irp,kplp,isp) = &
      vywf(ist,i,irp,kplp,isp) &
      + vylmd(m,i-l*l,0) * dwf(ist) &
      * rsymm%chi(irp,itran) * p_pot%ekbi(lp,ity)

  do ist = 1, eigencount
    vywf(ist,i,irp,kplp,isp) = &
    vywf(ist,i,irp,kplp,isp) &
    + vylmd(m,i-l*l,0) * dwf(ist) &
    * rsymm%chi(irp,itran) * p_pot%ekbi(lp,ity)

  div_proj(ist,i,irp,kplp,isp,1) = &
  div_proj(ist,i,irp,kplp,isp,1) &
  + vylmd(m,i-l*l,1) * dwf(ist) &
  * rsymm%chi(irp,itran)

...
Improved Non-local Force Calculation

```fortran
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 \( \Delta \)force = f(A&B)
        reduceAll(comm(atom), \( \Delta \)force)
        store locally with master of comm(atom)
    end atom
!$OMP END DO
end type
reduceAll(world, \( \Delta \)force)
do atom
    calc force = force + \( \Delta \)force 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)

Cori KNL: 16 nodes, 64 ranks, 17 threads/rank

THREAD_MULTIPLE OFF

forcnloc (363 sec)

THREAD_MULTIPLE ON

(113 sec)

Si$_{3917}$H$_{103}$
Non-Local Force Scaling Results

A, B and Δforce calcs vectorized and threaded.

Threading MPI_COMM_SPLIT. Currently being investigated for extremely large numbers of atoms.

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