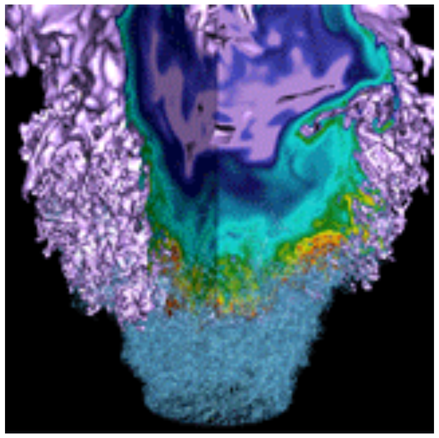
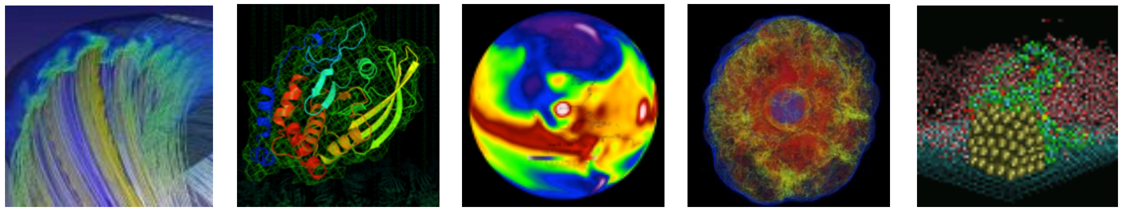
Performance Optimization of Quantum Espresso on KNL





Taylor Barnes, Thorsten Kurth, Paul Kent, Pierre Carrier, Nathan Wichmann, David Prendergast, Jack Deslippe

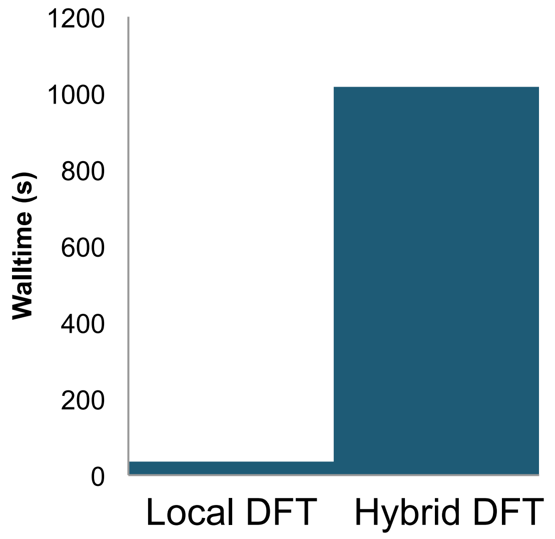
tbarnes@lbl.gov
NERSC
November 3 2016

Introduction

Goal: Prepare QE for large-scale execution on the KNL architecture, with a particular focus on improving the implementation of hybrid exchange

Cost of Hybrid DFT

Total walltime for an SCF calculation on 16 waters:



Local DFT:

$$\mathrm{K}\ \psi_i(x_1) = \overbrace{v_{\mathrm{xe}}[\rho(x_1)]}^{\text{Approximate exchange functional}}\ \psi_i(x_1)$$

Hybrid DFT:

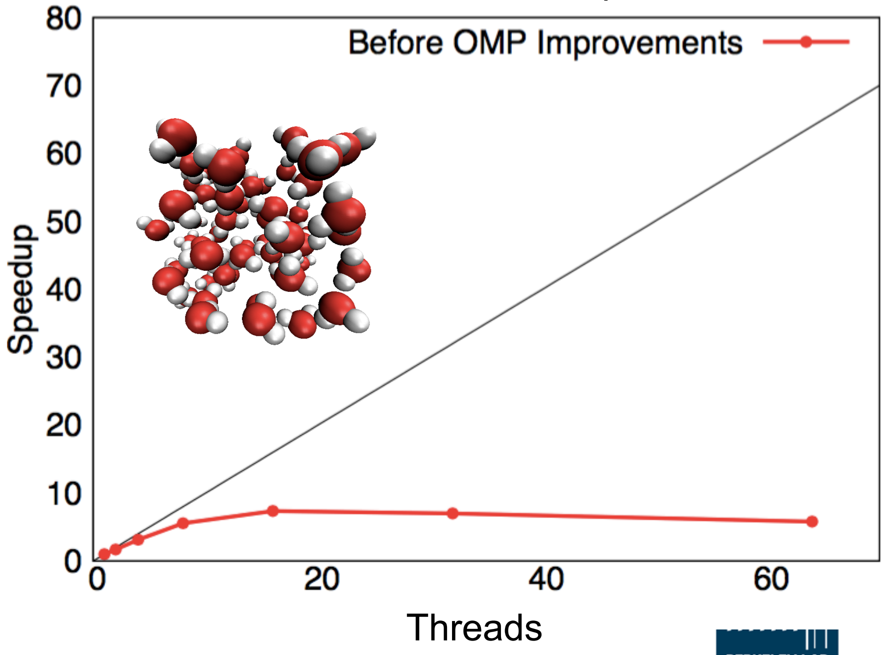
$$\mathrm{K}_j\,\psi_i(x_1) = \overbrace{\left[\int \frac{\psi_i(x_2)\,\psi_j(x_2)}{|x_2 - x_1|}\,d\,x_1\right]}^{\text{Exact exchange operator}}\psi_j(x_1)$$

Original OpenMP Threading

```
procedure VEXX
    ...
    for $i$ in 1:$n_{occ}$ do
        ...
        $c(\mathbf{g}) = \text{FFT}[1/|\mathbf{r}' - \mathbf{r}|]$
        for $j$ in 1:$n_{occ}$ do
            $\rho_{ij}(\mathbf{r}) = \psi_i(\mathbf{r})\psi_j(\mathbf{r})$
            $\rho_{ij}(\mathbf{g}) = \text{FFT}[\rho_{ij}(\mathbf{r})]$
            $v_{ij}(\mathbf{g}) = c(\mathbf{g})\rho_{ij}(\mathbf{g})$
            $v_{ij}(\mathbf{r}) = \text{FFT}^{-1}[v_{ij}(\mathbf{g})]$
            $(K\psi_i)(\mathbf{r})$ += $\psi_j(\mathbf{r})v_{ij}(\mathbf{r})$
```

```
DO j=1,n_occ
!$omp parallel do ...
   DO ir = 1, nr
      rho(ir)=psi(ir,i)*psi(ir,j)
   ENDDO
!$omp end parallel do
   ...
ENDDO
```

NERSC Whitebox, Quad Cache Mode;
Intel 17.0.042 Compiler:

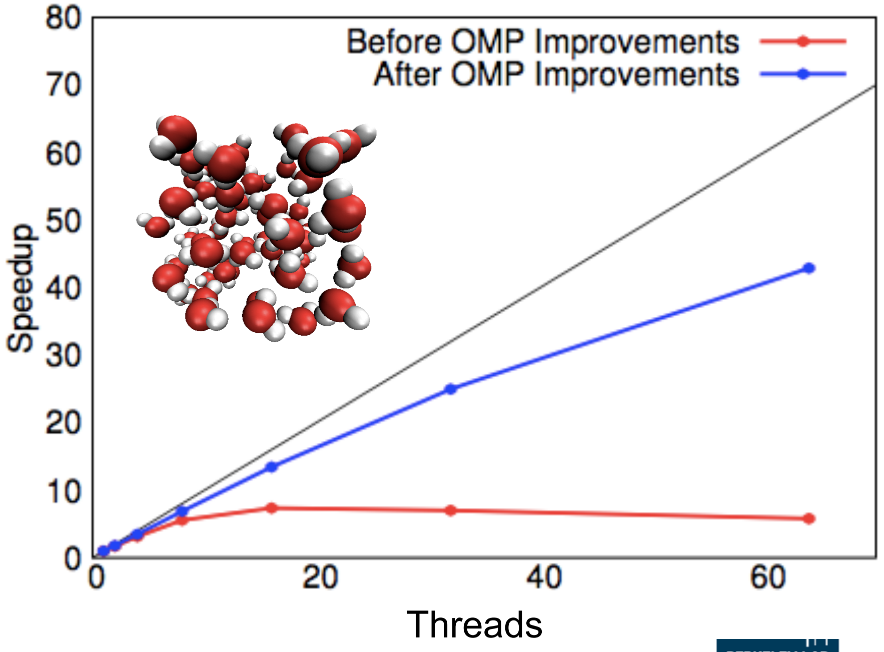


Improved OpenMP Threading

```
procedure VEXX
    ...
    for $i$ in 1:$n_{occ}$ do
        ...
        $c(\mathbf{g}) = \mathrm{FFT}[1/|\mathbf{r}' - \mathbf{r}|]$
        for $j$ in 1:$n_{occ}$ do
            $\rho_{ij}(\mathbf{r}) = \psi_i(\mathbf{r})\psi_j(\mathbf{r})$
            $\rho_{ij}(\mathbf{g}) = \mathrm{FFT}[\rho_{ij}(\mathbf{r})]$
            $v_{ij}(\mathbf{g}) = c(\mathbf{g})\rho_{ij}(\mathbf{g})$
            $v_{ij}(\mathbf{r}) = \mathrm{FFT}^{-1}[v_{ij}(\mathbf{g})]$
            $(K\psi_i)(\mathbf{r}) \mathrel{+}= \psi_j(\mathbf{r})v_{ij}(\mathbf{r})$
```

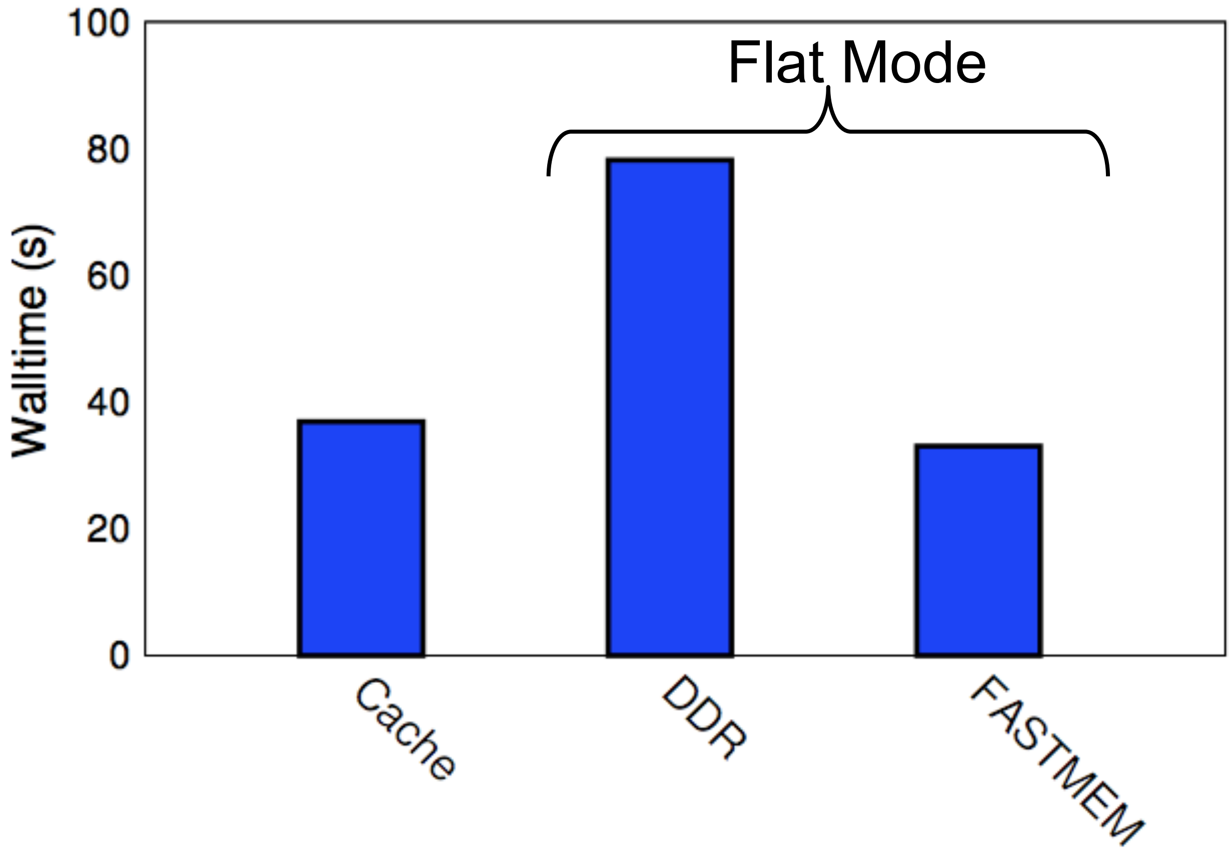
```
!$omp parallel do ...
DO j=1, n_occ
   DO ir = 1, nr
      rho(ir,j)=psi(ir,i)*psi(ir,j)
   ENDDO
ENDDO
!$omp end parallel do
```

NERSC Whitebox, Quad Cache Mode;
Intel 17.0.042 Compiler:



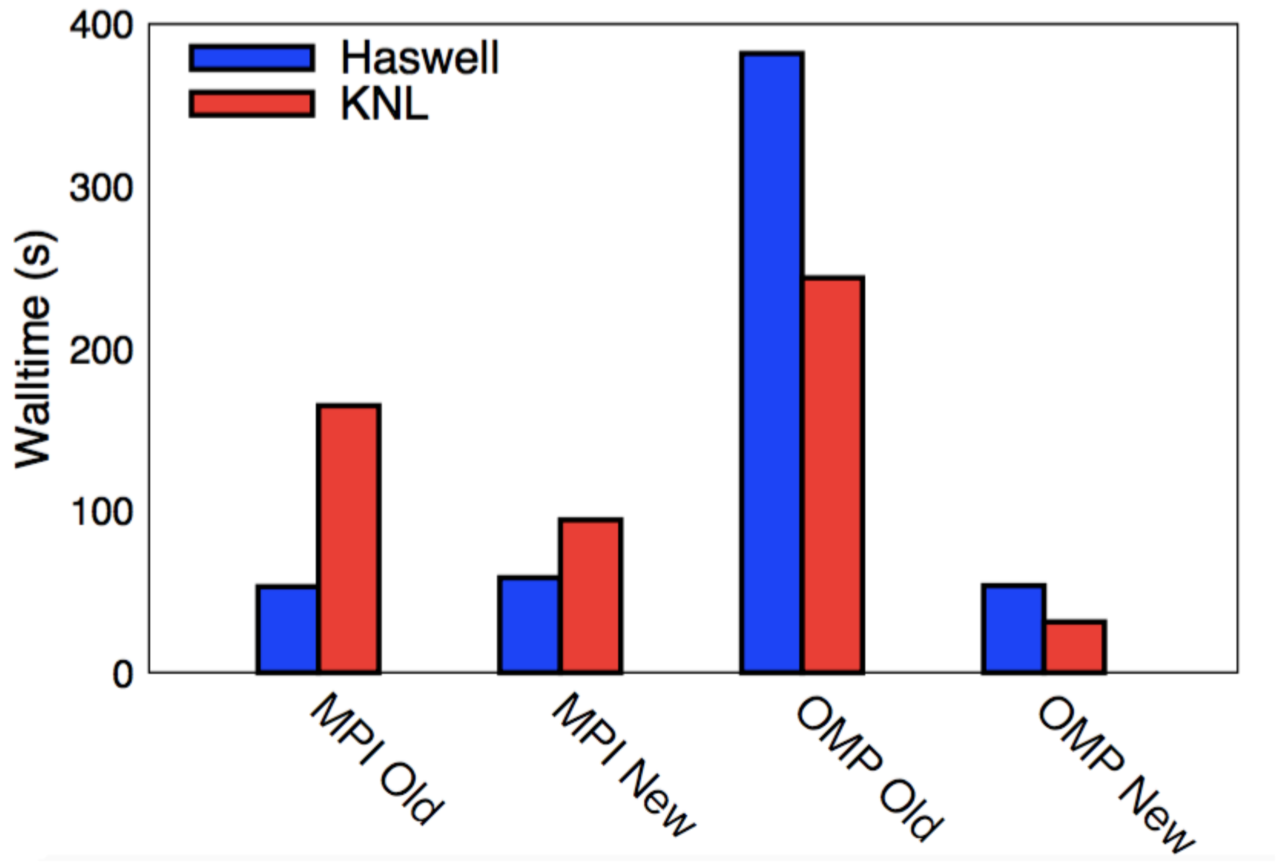
Increasing the amount of work performed in each OMP loop dramatically reduces overhead costs.

KNL NUMA Mode Performance



With the original code, running KNL in Cache mode is approximately 2x faster than Flat mode. Using FASTMEM directives enables Flat mode to outperform Cache mode.

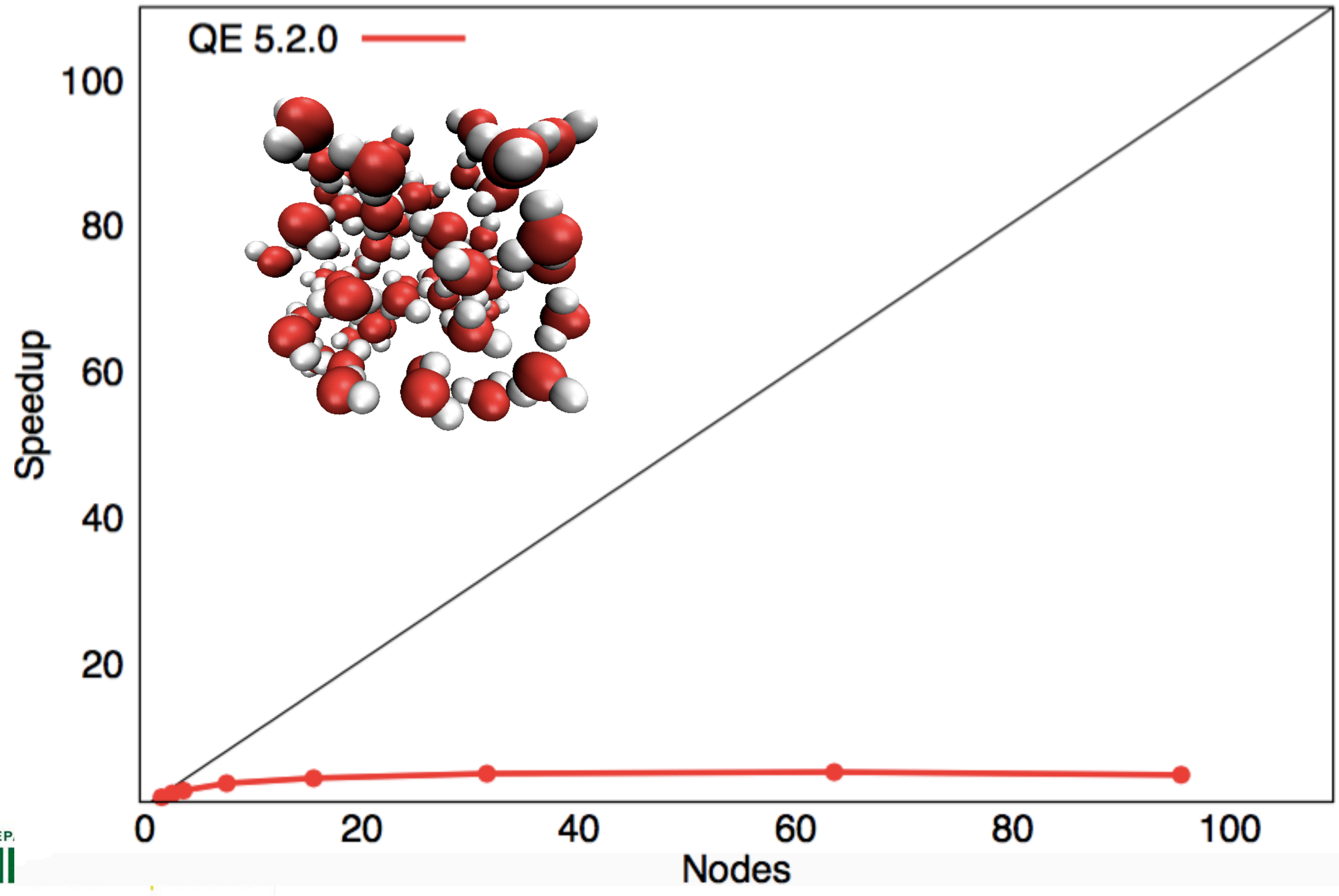
KNL vs. Haswell



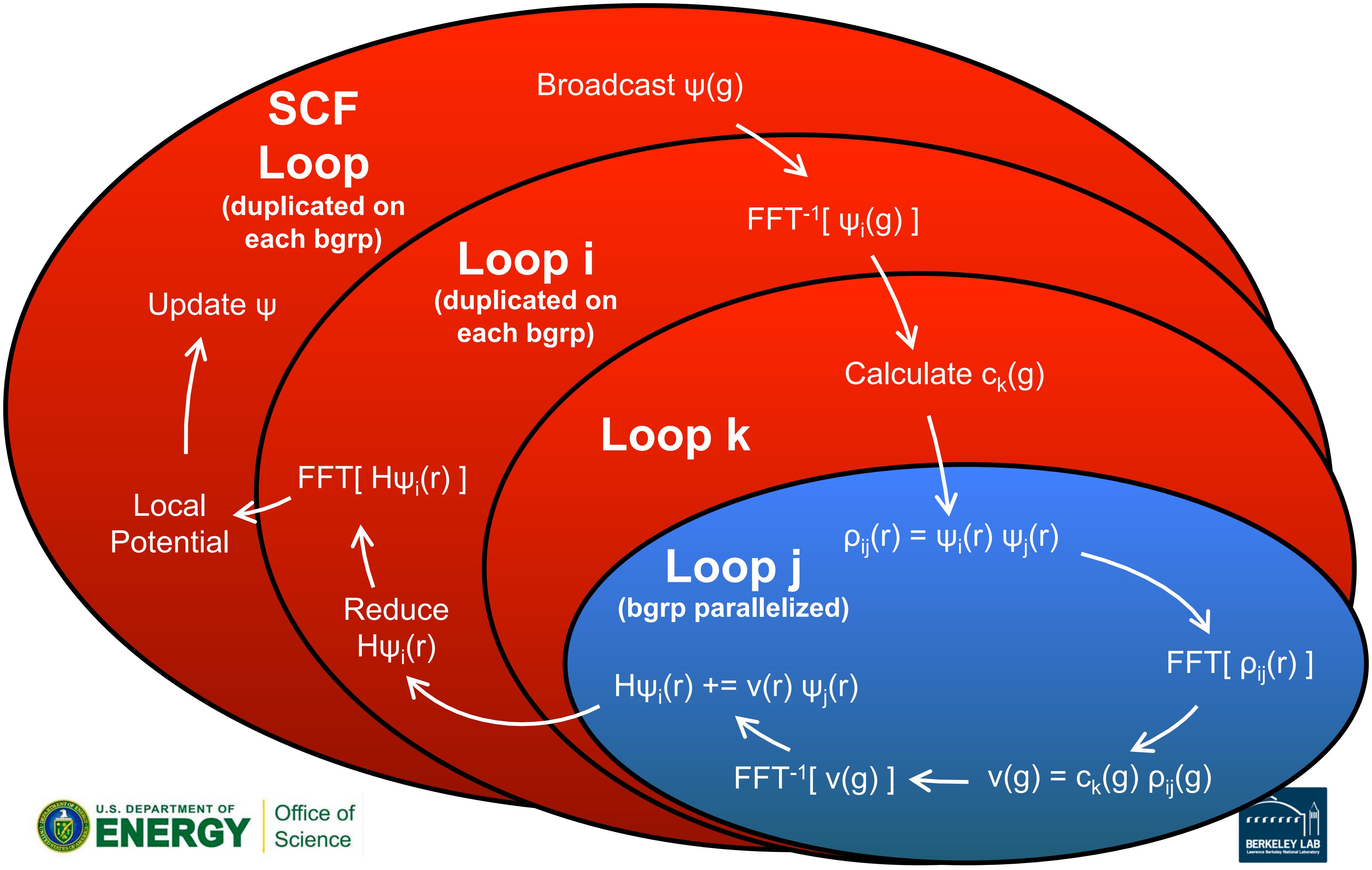
Using the improved OpenMP code is the fastest way to run on KNL, and is about 1.6x faster than the best Haswell walltimes. OMP is significantly improved on both Haswell and KNL.

Strong Scaling of QE

Strong scaling of QE on Ivy Bridge, using pure MPI mode:

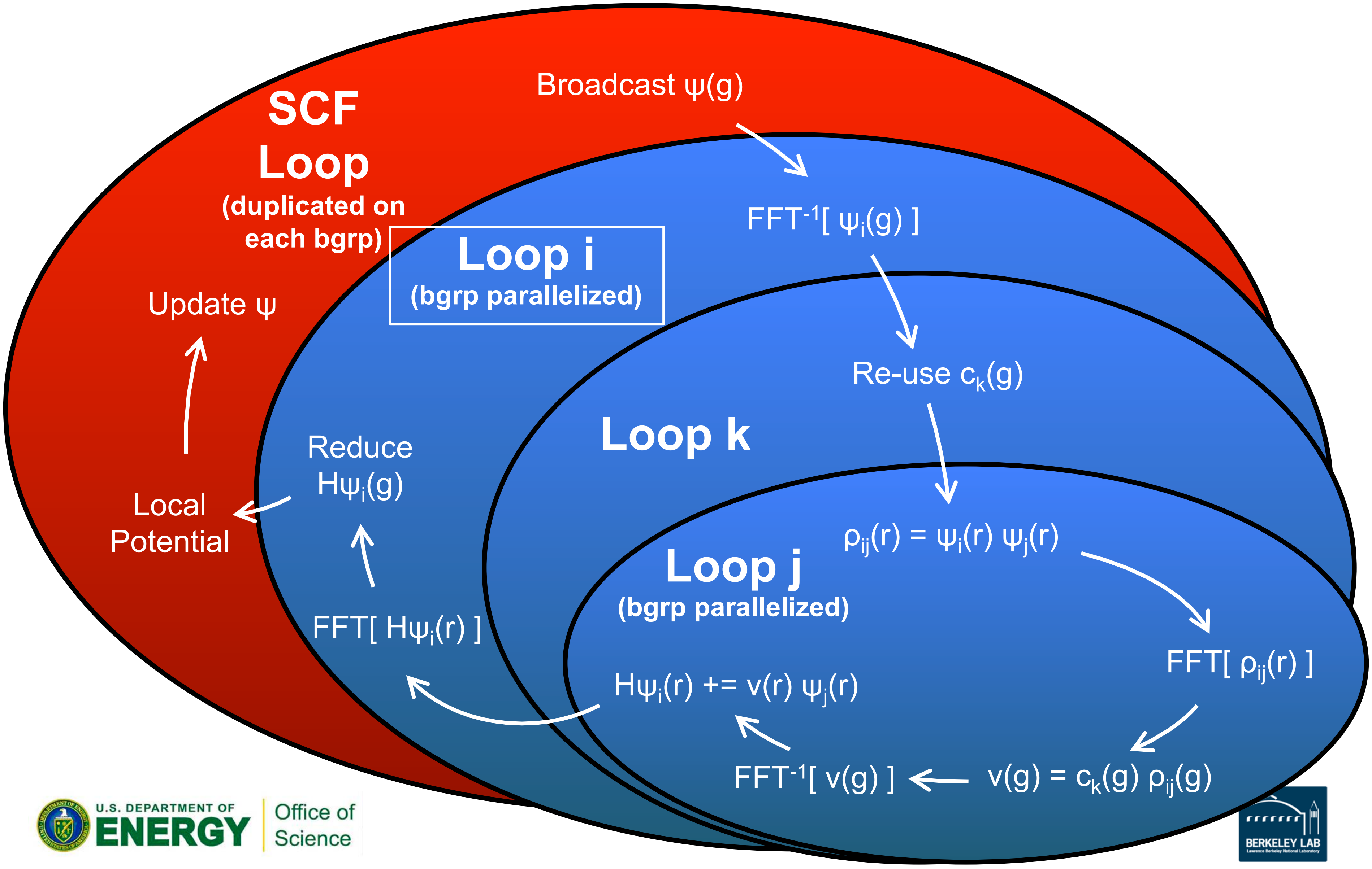


Overview of Existing Code



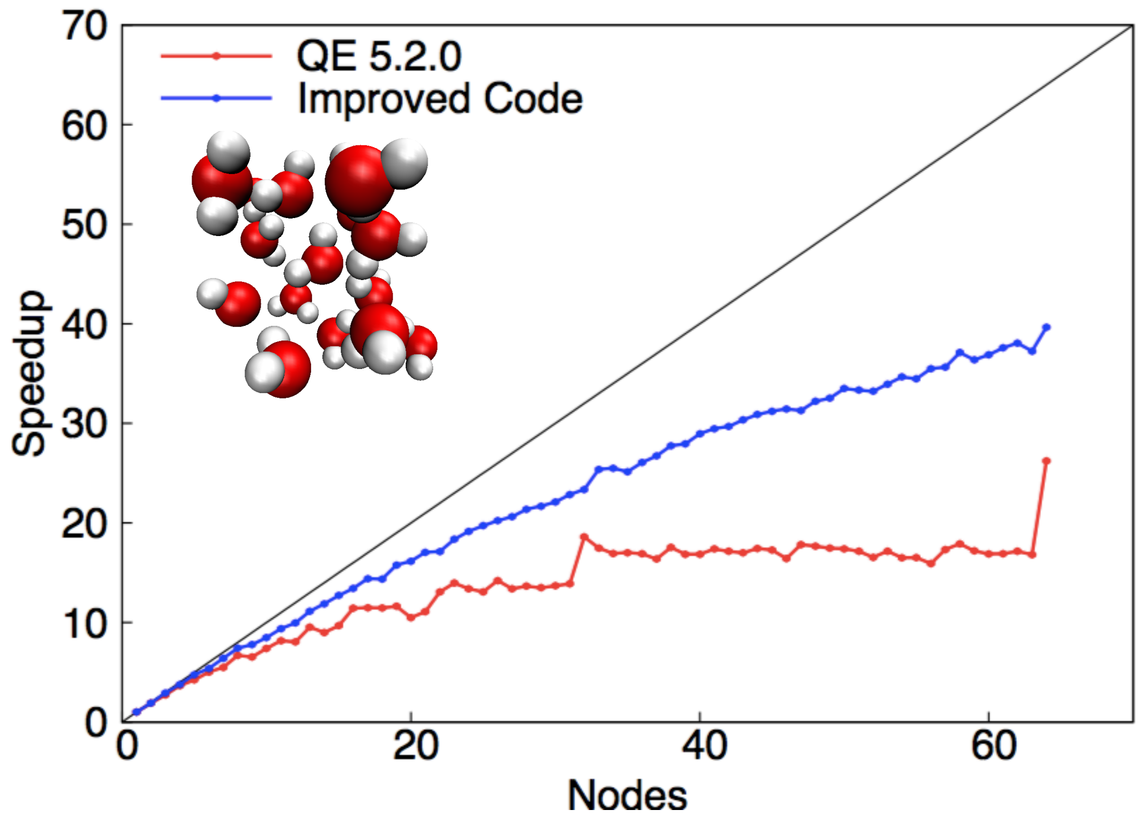
SCF Loop (duplicated on each bgrp)
Broadcast ψ(g)
Update ψ
Local Potential
Loop i (duplicated on each bgrp)
$FFT^{-1}[\ \psi_i(g)\ ]$
$FFT[\ H\psi_i(r)\ ]$
Reduce $H\psi_i(r)$
Loop k
Calculate $c_k(g)$
Loop j (bgrp parallelized)
$\rho_{ij}(r) = \psi_i(r)\ \psi_j(r)$
$FFT[\ \rho_{ij}(r)\ ]$
$v(g) = c_k(g)\ \rho_{ij}(g)$
$FFT^{-1}[\ v(g)\ ]$
$H\psi_i(r) \mathrel{+}= v(r)\ \psi_j(r)$

Pair Parallelization


SCF Loop (duplicated on each bgrp)
Broadcast ψ(g)
Loop i (bgrp parallelized)
FFT^{-1}[$\psi_i(g)$]
Re-use $c_k(g)$
Loop k
Update ψ
Reduce $H\psi_i(g)$
Local Potential
$\rho_{ij}(r) = \psi_i(r)\ \psi_j(r)$
Loop j (bgrp parallelized)
FFT[$H\psi_i(r)$]
FFT[$\rho_{ij}(r)$]
$H\psi_i(r)$ += $v(r)\ \psi_j(r)$
FFT^{-1}[$v(g)$]
$v(g) = c_k(g)\ \rho_{ij}(g)$

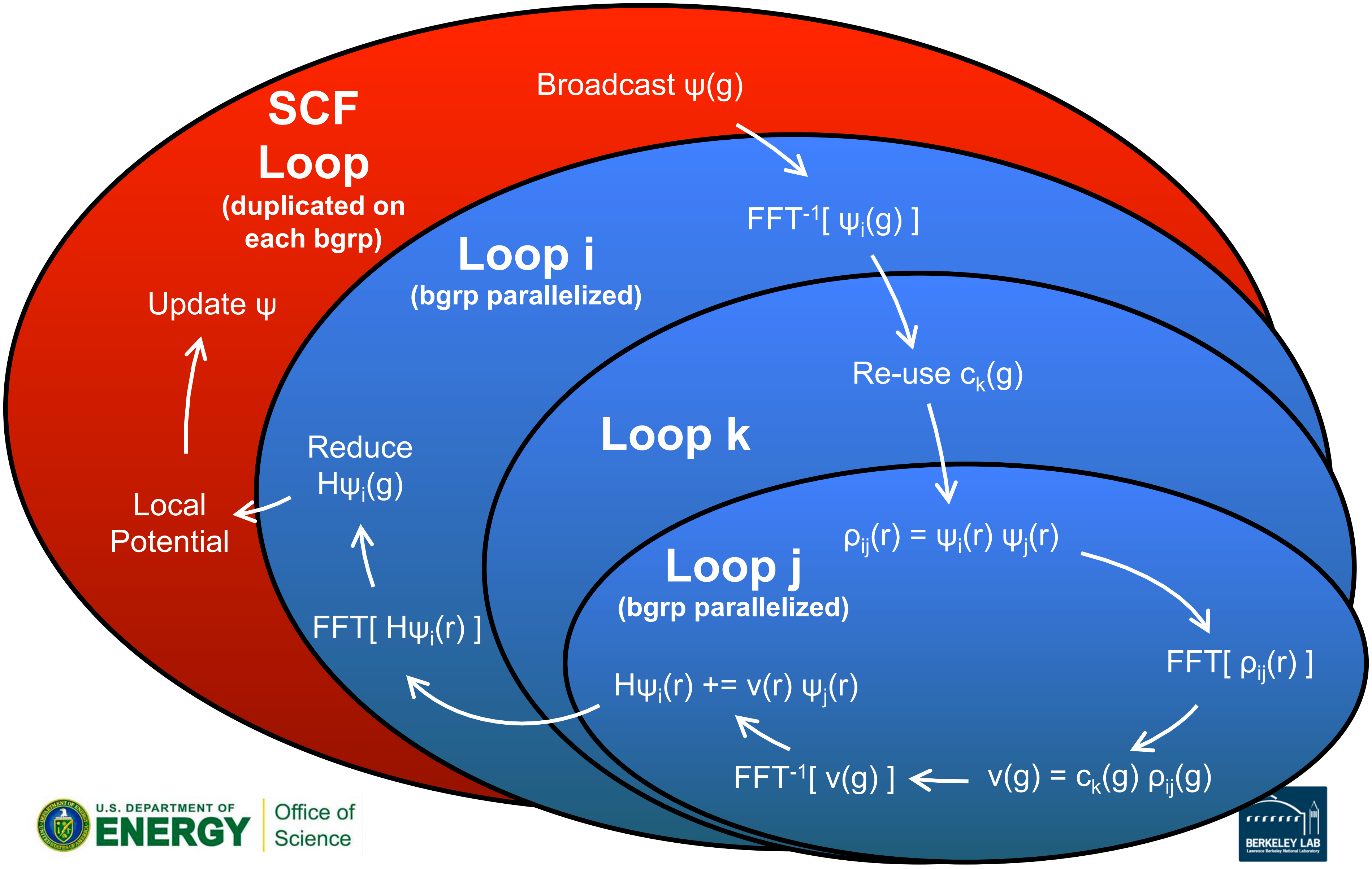
Pair Parallelization Performance

Strong scaling of the exact exchange part of the code on Ivy Bridge, with 1 band group per node:



Parallelization over band pairs both improves the strong scaling of the code and also improves the load balancing.

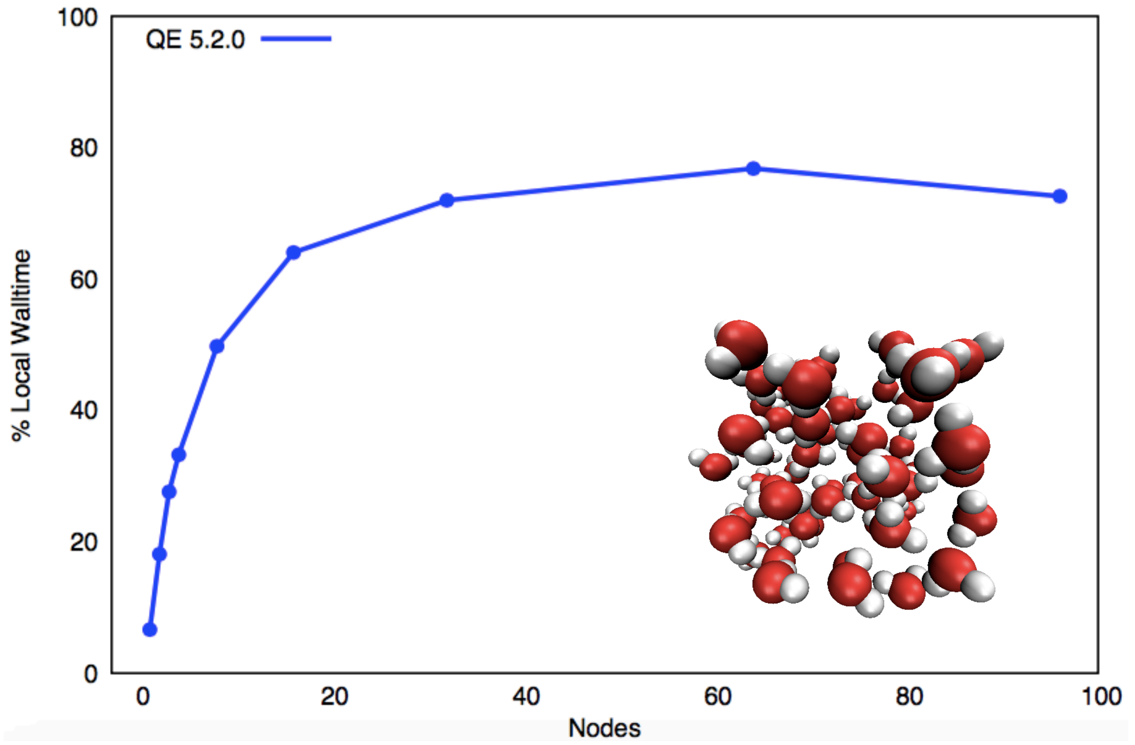
Code Overview



SCF Loop (duplicated on each bgrp)
Broadcast ψ(g)
Update ψ
Local Potential
Loop i (bgrp parallelized)
$FFT^{-1}[\ \psi_i(g)\ ]$
Reduce $H\psi_i(g)$
$FFT[\ H\psi_i(r)\ ]$
Loop k
Re-use $c_k(g)$
Loop j (bgrp parallelized)
$\rho_{ij}(r) = \psi_i(r)\ \psi_j(r)$
$FFT[\ \rho_{ij}(r)\ ]$
$v(g) = c_k(g)\ \rho_{ij}(g)$
$FFT^{-1}[\ v(g)\ ]$
$H\psi_i(r) \mathrel{+}= v(r)\ \psi_j(r)$

Cost of the Local Calculation

Fraction of total walltime spent in local regions of the code, using 1 band group per node on Haswell:

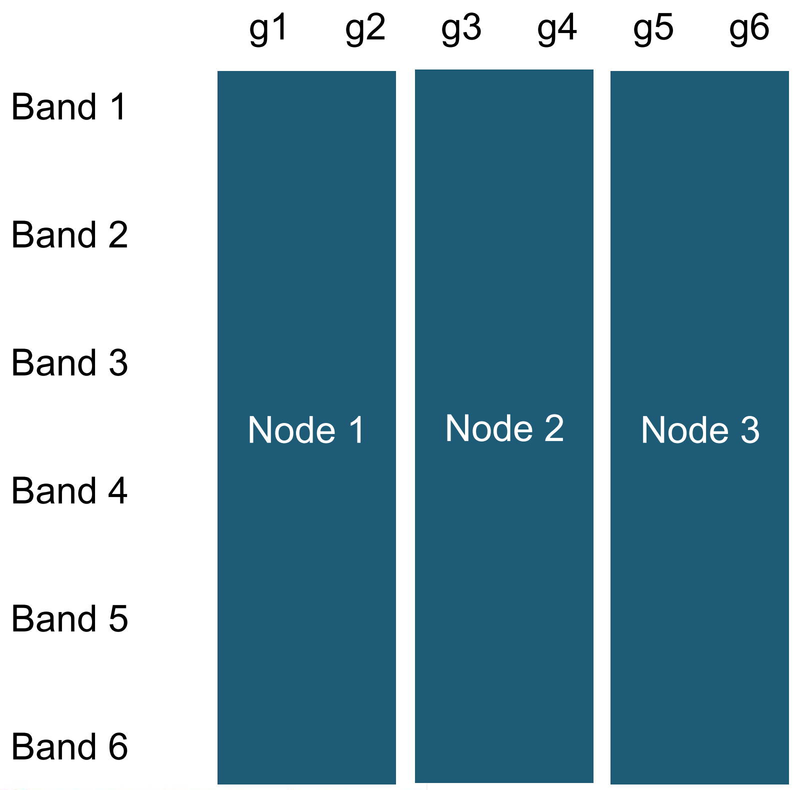


Unintuitively, local regions of the code dominate the cost of the calculation when using large numbers of nodes. This is because the local regions of the code a run in serial with respect to band groups.

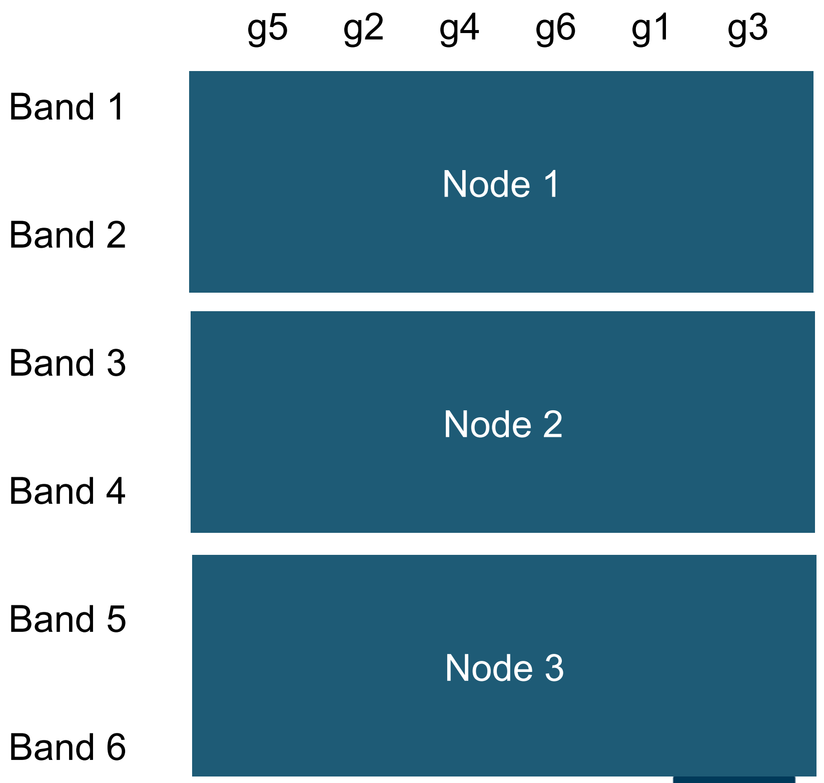
Independent Parallelization of the Local Code

Enabling parallelization of the local regions of the code across band groups requires on-the-fly transformation of the data structures between local and exact exchange regions of the code.

Data structure of Ψ(g), in local code:

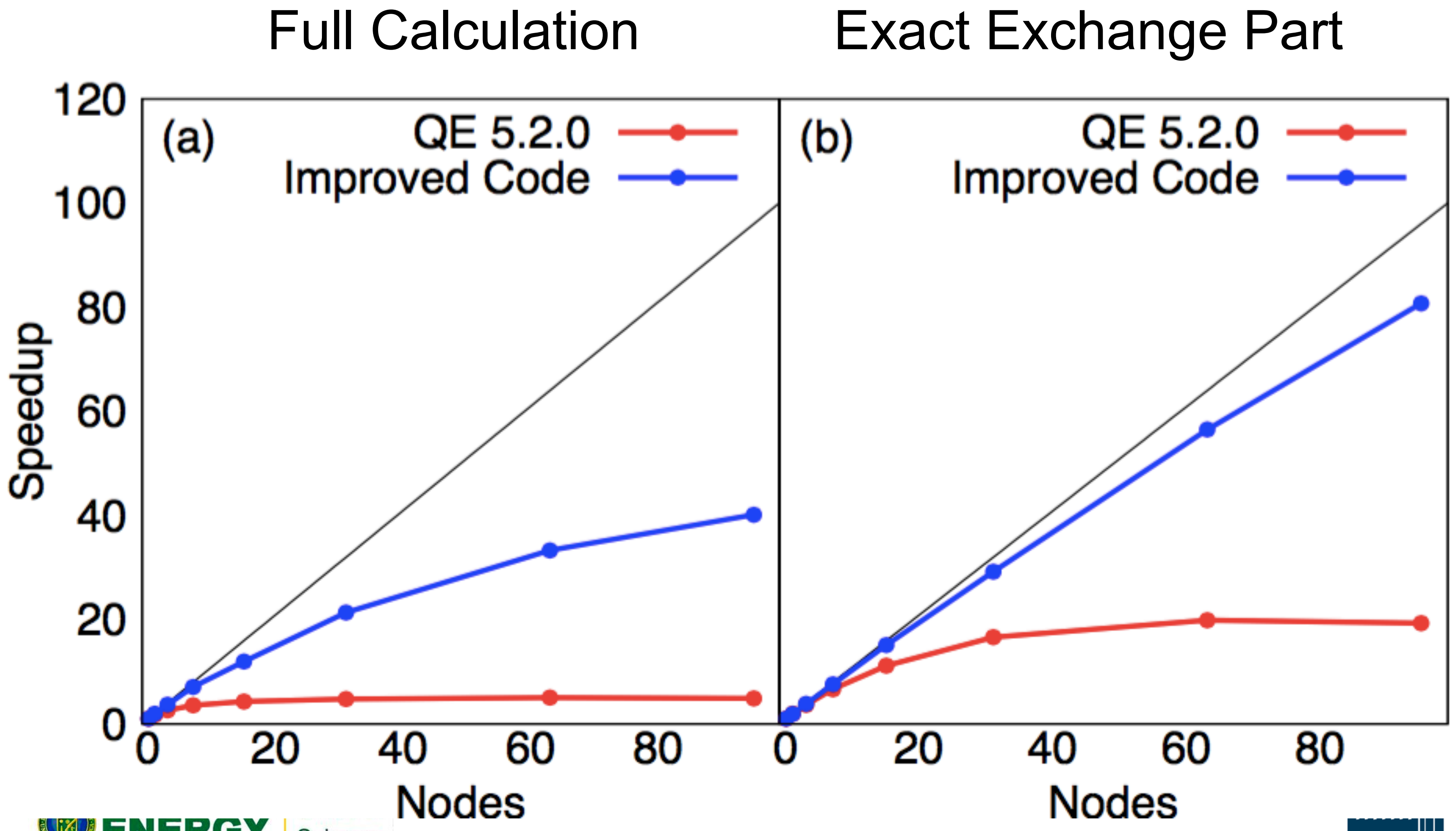


Data structure of Ψ(g), in exact exchange code:



Improved Strong Scaling

Strong scaling on Ivy Bridge, with 1 band group per node:



Thank You