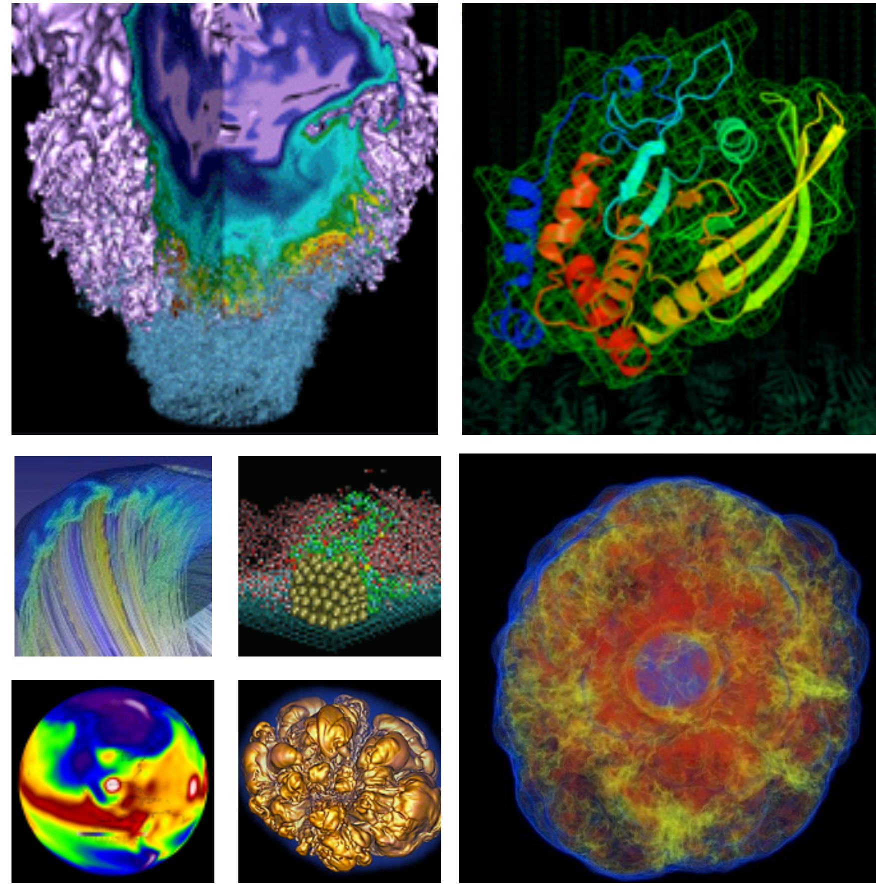
Optimizing EXX Performance on Intel Xeon Phi



Thorsten Kurth

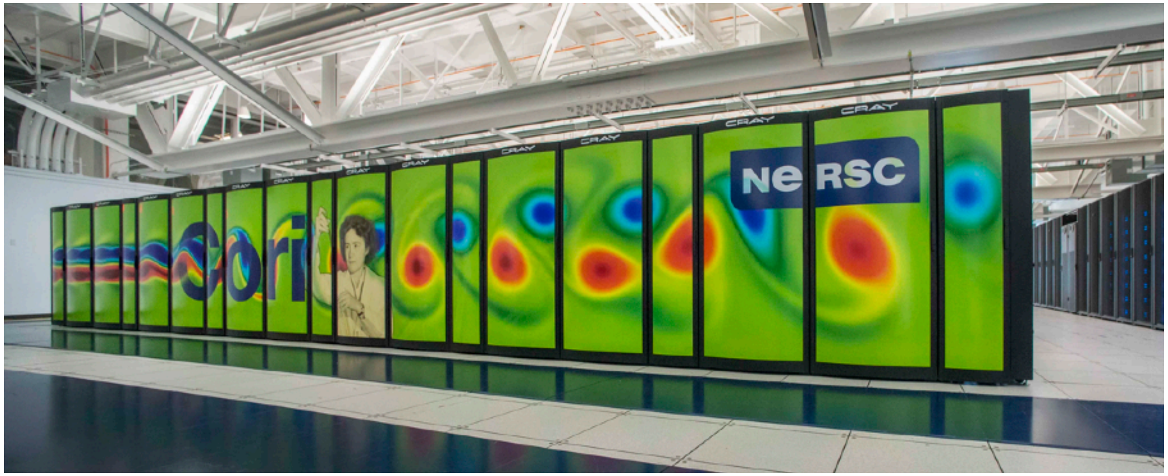
QE Developers Workshop,
10.1.2017

Xeon Phi 7250 Features

- important hardware features
  - 68 cores@1.2 Ghz, 272 threads in total
  - complicated memory topology: configurable 2D on-chip interconnect, shared L2 cache per tile
  - 512bit-wide vector units with FMA support and additional fast reduced precision intrinsics
  - 16 GB high-bandwidth on-package memory (HBM/ MCDRAM), configurable as cache, flat or hybrid cache/flat
- KNL-ready applications should exploit some of these features

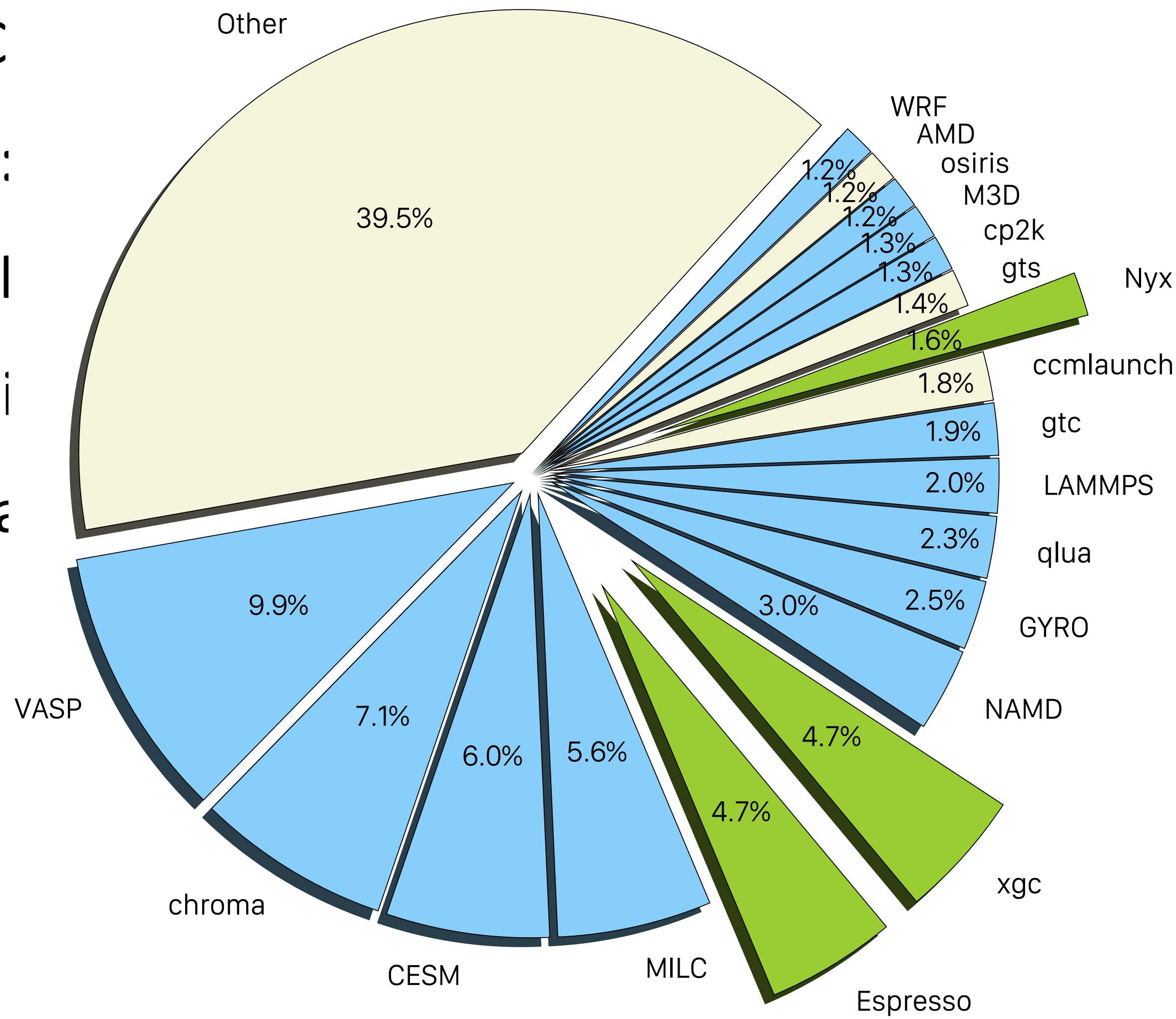
Cori System Description

- Cray XC40 supercomputer
- Phase I: 2004 Haswell nodes, 1.92 PFlops/sec
- Phase II: 9304 Xeon Phi 7250 nodes, 27.9 PFlops/sec
- Cray Aries high-speed interconnect with Dragonfly topology
- Aggregate memory: 203 TB (Phase I), 1 PB (Phase II)



Cori System Description

- Cray XC
- Phase I:
- Phase II
- Cray Ari   topology
- Aggrega   )



Introducing OpenMP

- focus on exact-exchange calculation (vexx)

$$(\hat{K}\psi_i)(\mathbf{r}) = -\sum_{j=1}^{n} \psi_j(\mathbf{r}) \int \mathrm{d}^3r' \frac{\psi_j^*(\mathbf{r}')\psi_i(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}$$

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

Introducing OpenMP

NERSC

- focus on exact-exchange calculation (vexx)

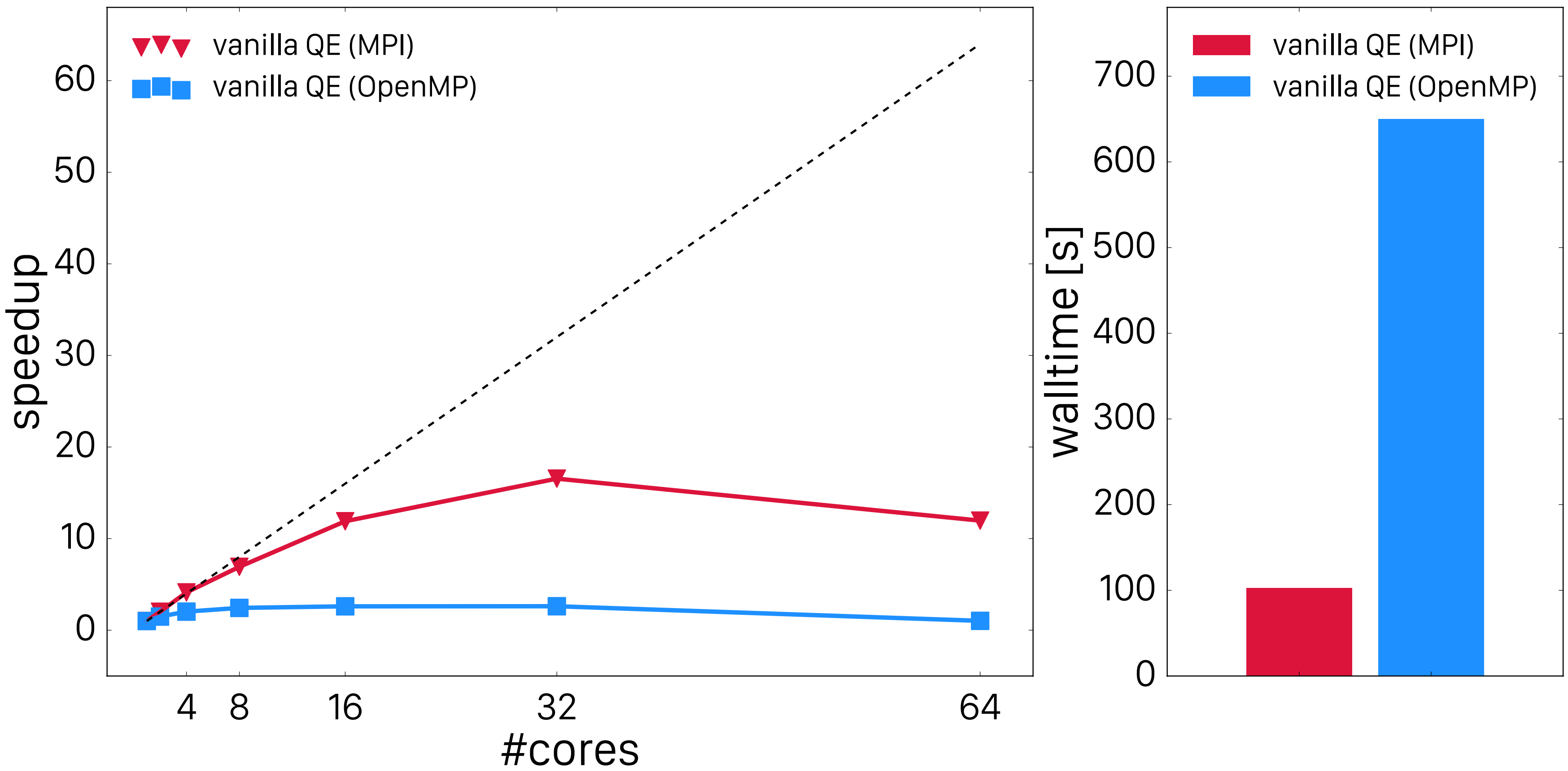
$$(\hat{K}\psi_i)(\mathbf{r}) = -\sum_{j=1}^{n} \psi_j(\mathbf{r}) \int \mathrm{d}^3 r' \frac{\psi_j^*(\mathbf{r}')\psi_i(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}$$

- first step: use OpenMP do speed-up scalar-products

```
procedure VEXX
    ...
    for i in 1:n do
        ...
        c(g) = FFT[1/|r' - r|]
        for j in 1:n do
            ρ_ij(r) = ψ_j^*(r) ψ_i(r)
            ρ_ij(g) = FFT[ρ_ij(r)]
            v_ij(g) = c(g) ρ_ij(g)
            v_ij(r) = FFT^{-1}[v_ij(g)]
            (K̂ψ_i)(r) += ψ_j(r) v_ij(r)
```

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

Baseline Results (16 H_2O)



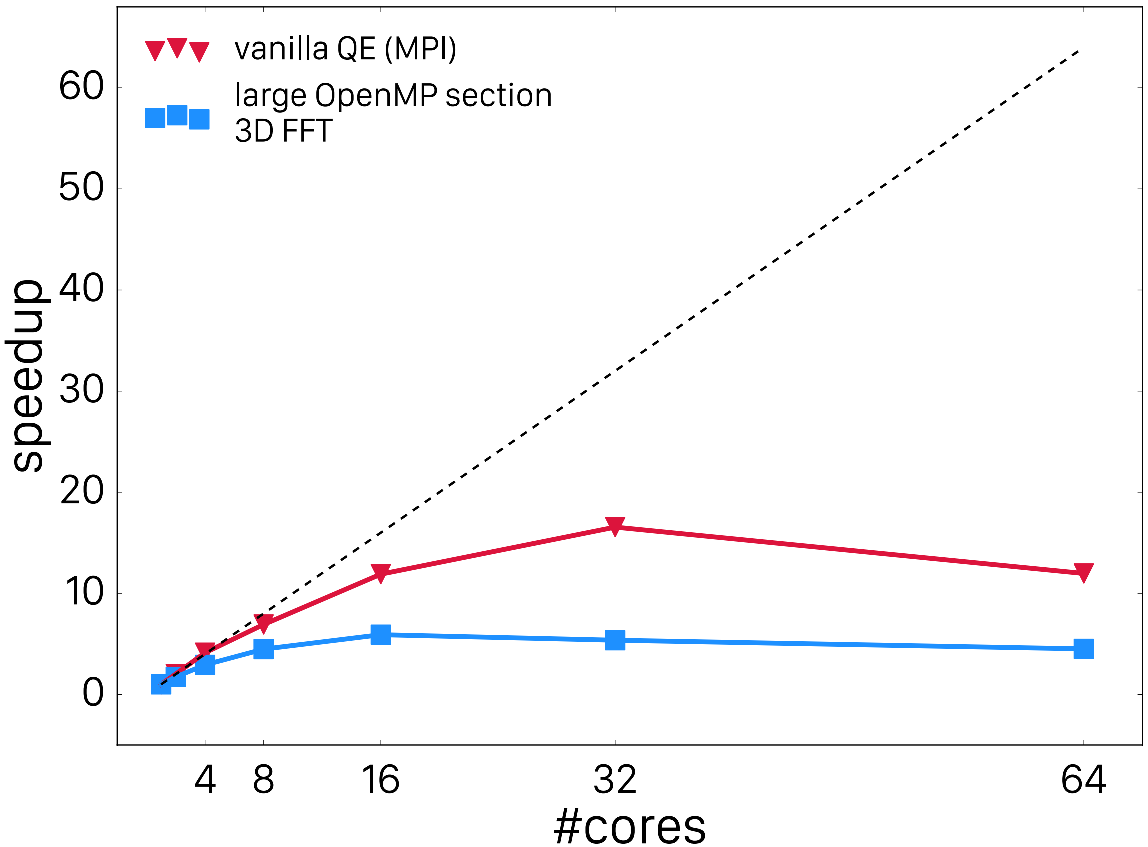
Reduce Fork-Join Operations

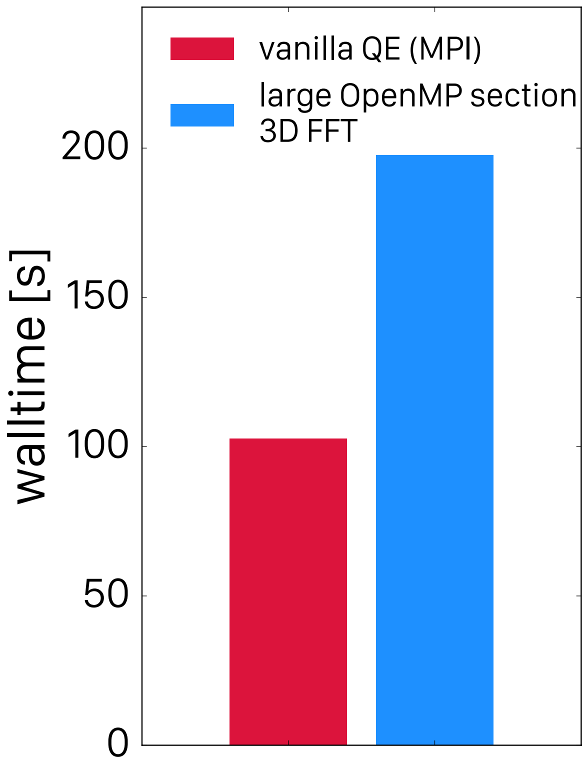
- move OpenMP into the outer loops to create enough work for threads

```
DO j=1:n
    ...
    DO ir=1:nr
        ...
    ENDDO
    ...
    FFT[...]
    ...
    DO ir=1:nr
        ...
    ENDDO
    ...
    IFFT[...]
    ...
ENDDO
```

```
DO j=1:n
    ...
    DO ir=1:nr
        ...
    ENDDO
    ...
ENDDO
DO j=1:n
    FFT3D[...]
ENDDO
DO j=1:n
    ...
    DO ir=1:nr
        ...
    ENDDO
    ...
ENDDO
DO j=1:n
    IFFT3D[...]
ENDDO
DO j=1:n
    ...
ENDDO
```

Large OpenMP Sections (16 H_2O)





Tile Loops

- problem: with `collapse(2)`, many scatter-gather instructions generated although data access is contiguous (compiler bug?)

Tile Loops

- problem: with `collapse(2)`, many scatter-gather instructions generated although data access is contiguous (compiler bug?)
- solution: manual blocking/tiling

```
nblock = 2048
nrt = nr / nblock
!$omp parallel do collapse(2)…
DO ist = 1, nrt
     DO j=1, n
          ir_start = (irt - 1) * nblock + 1
          ir_end = min(ir_start+nblock-1,nr)
          DO ir = ir_start, it_end
               rho(ir,j)=conjg(psi(ir,j)) * psi(ir,i)
          ENDDO
     ENDDO
ENDDO
!$omp end parallel do
```

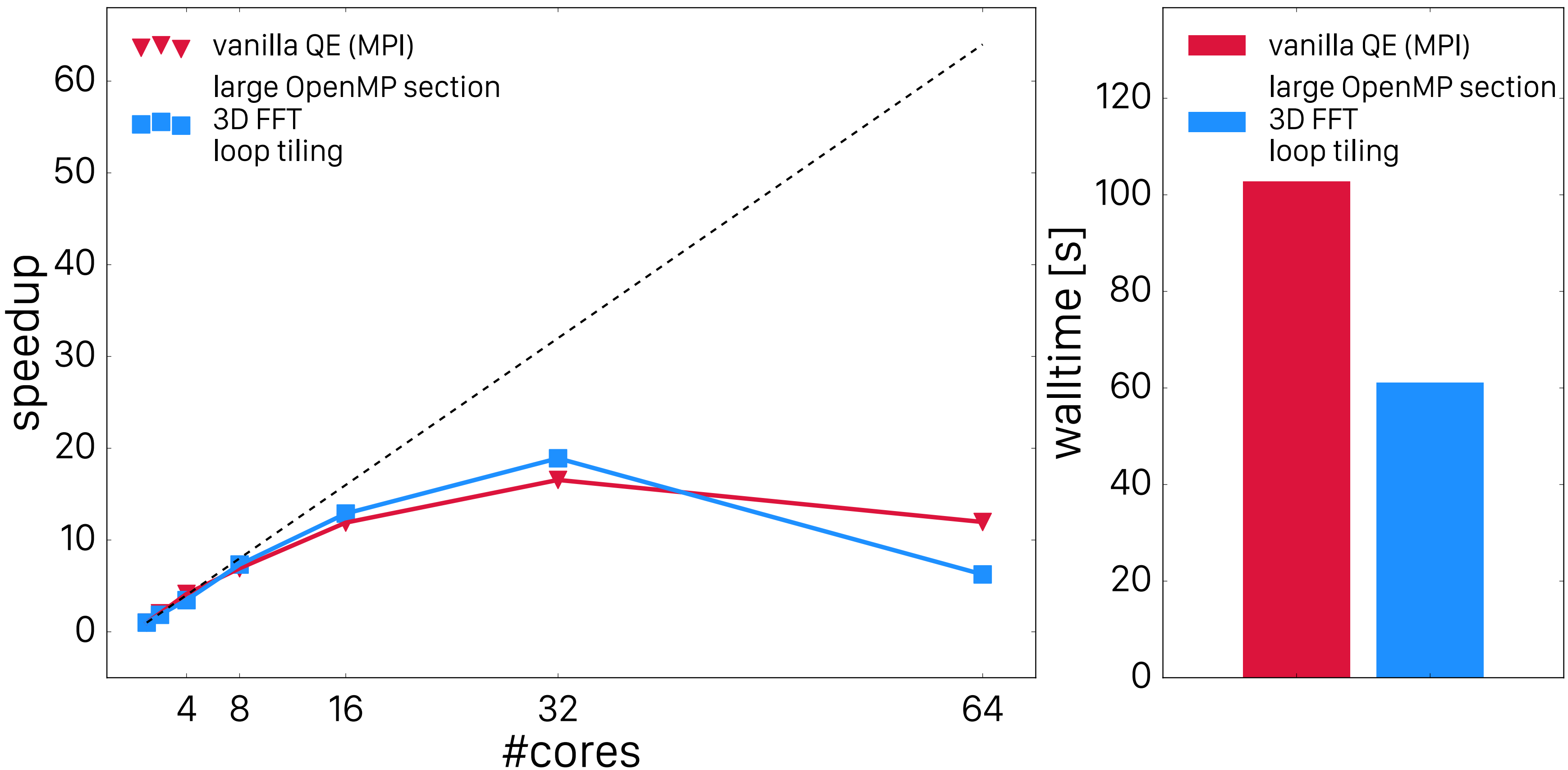
Tile Loops

- problem: with `collapse(2)`, many scatter-gather instructions generated although data access is contiguous (compiler bug?)
- solution: manual blocking/tiling

```
nblock = 2048
nrt = nr / nblock
!$omp parallel do collapse(2)…
DO ist = 1, nrt
     DO j=1, n
          ir_start = (irt - 1) * nblock + 1
          ir_end = min(ir_start+nblock-1,nr)
          DO ir = ir_start, it_end
               rho(ir,j)=conjg(psi(ir,j)) * psi(ir,i)
          ENDDO
     ENDDO
ENDDO
!$omp end parallel do
```

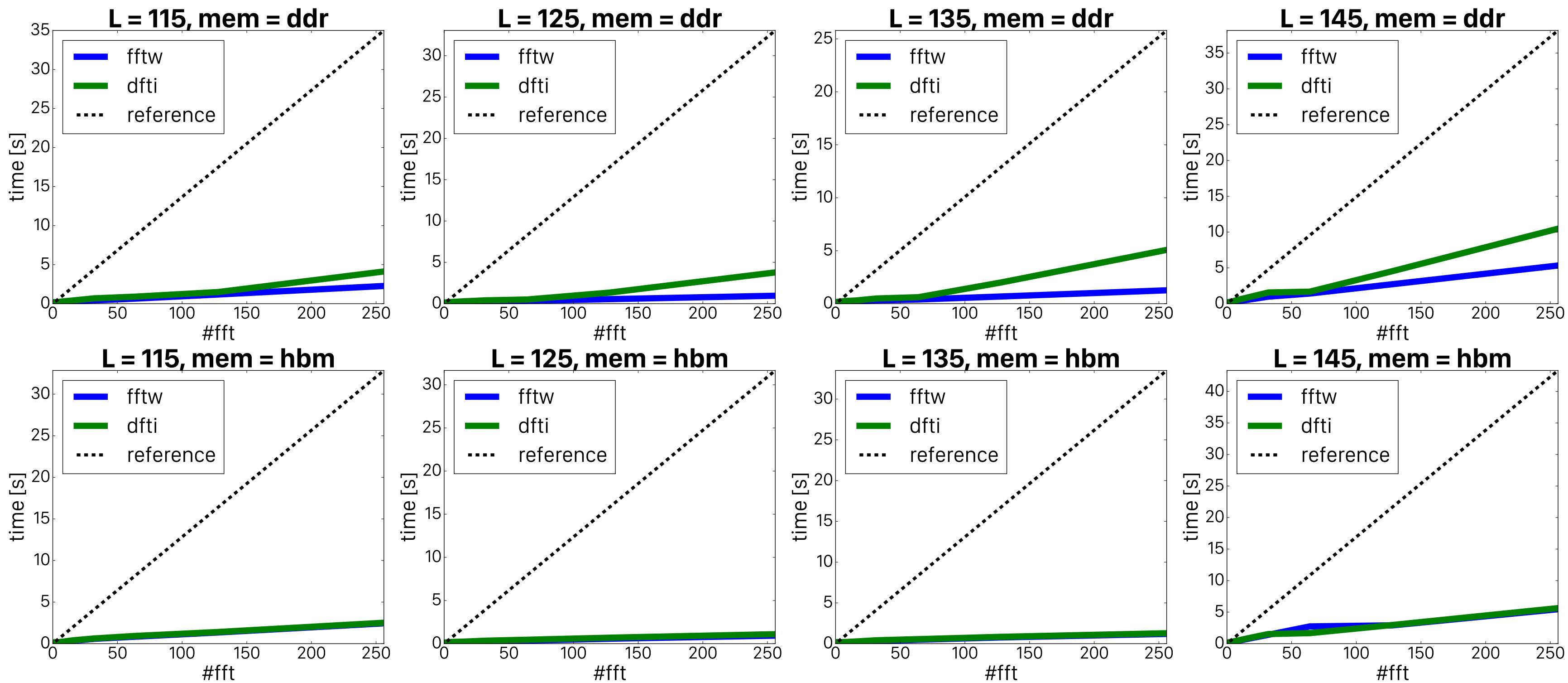
- future improvement: factor these routines into subroutines to hide code complexity

Large Sections + Tiling (16 H_2O)

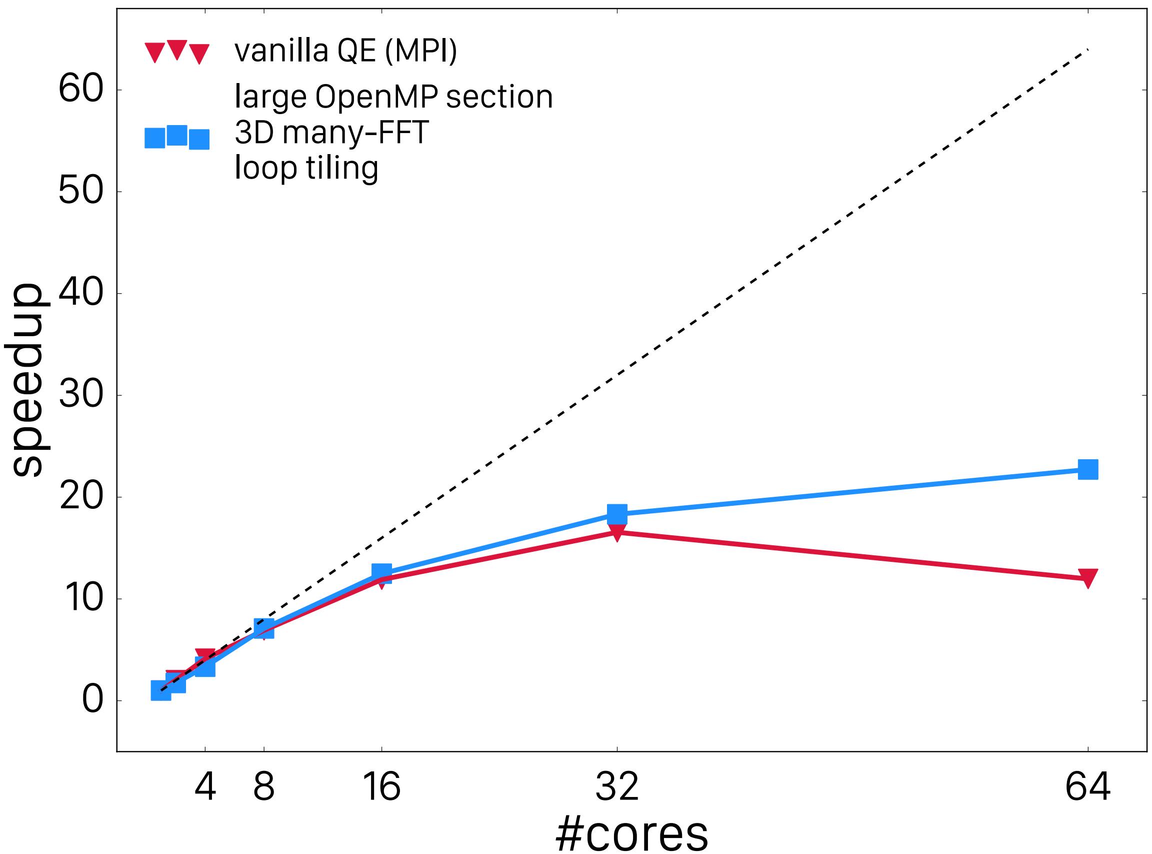


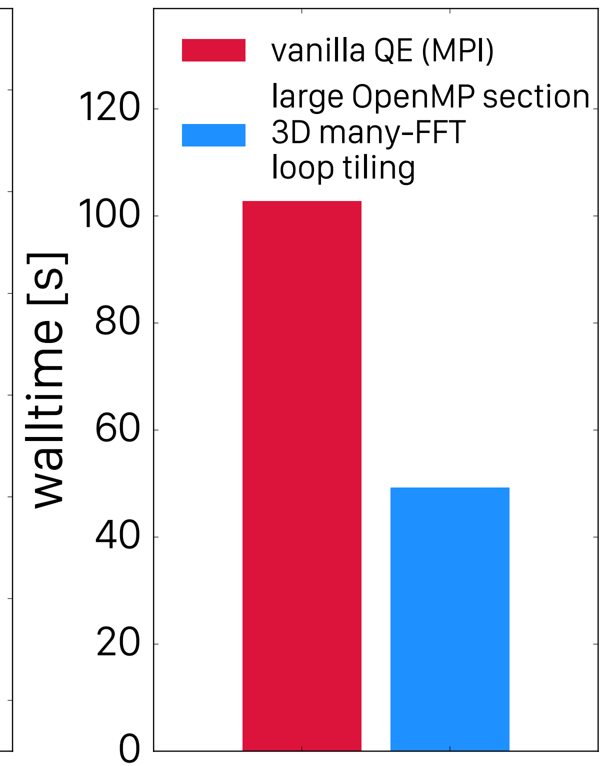
Batch-Process FFTs

- pool FFTs in order to perform multiple at once (reorganize: 3D FFTs can be performed locally)



Large Sections + Tiling + FFT-Batching (16 H_2O)





Summary

- improved code shows significant better OpenMP scaling (effect bigger for bigger systems)
- outperforms all-MPI mode
- EXX part is now subdominant, especially when ACE is used
- need to work on the PBE part, especially the diagonalization

NERSC

Thank you