IXPUG FALL CONFERENCE 2018

IMPROVED THREADING PERFORMANCE OF QUANTUM ESPRESSO

YE LUO
Assistant Scientific Applications Engineer
Computational Science Division,
Argonne National Laboratory

Sep 26th 2018, Hillsboro, Oregon

OUTLINE

- Motivation
  - What is Quantum ESPRESSO
  - Why threading is important
- How to improve the threading of QE
  - 3D parallel FFT
  - Davidson solver
- Results
  - Strong scaling
  - Time to solution

Acknowledgments:

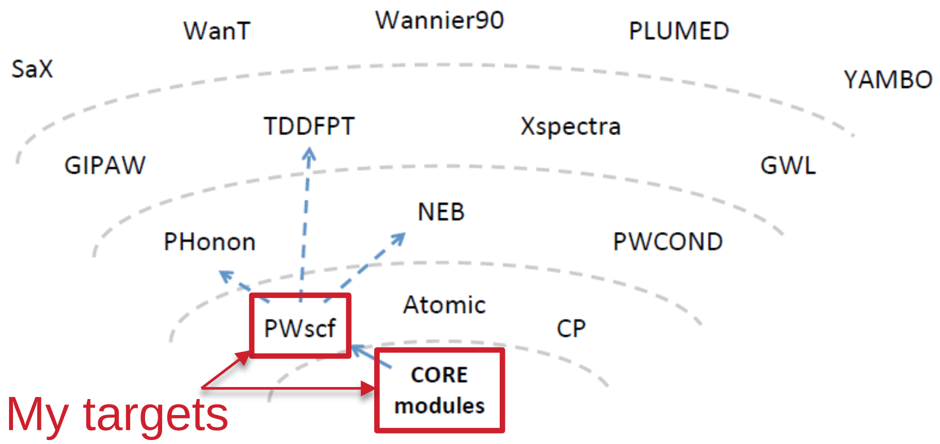
- This research used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility operated undercontract DE-AC02-06CH11357
- We gratefully acknowledge the computing resources provided on Bebop, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

QUANTUM ESPRESSO

www.quantum-espresso.org

- MANIFESTO: QUANTUM ESPRESSO is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.
- The 2009 published paper gets over 11k citations.
- Widely used for high throughput material research in US, EU, China.

QE package portfolio

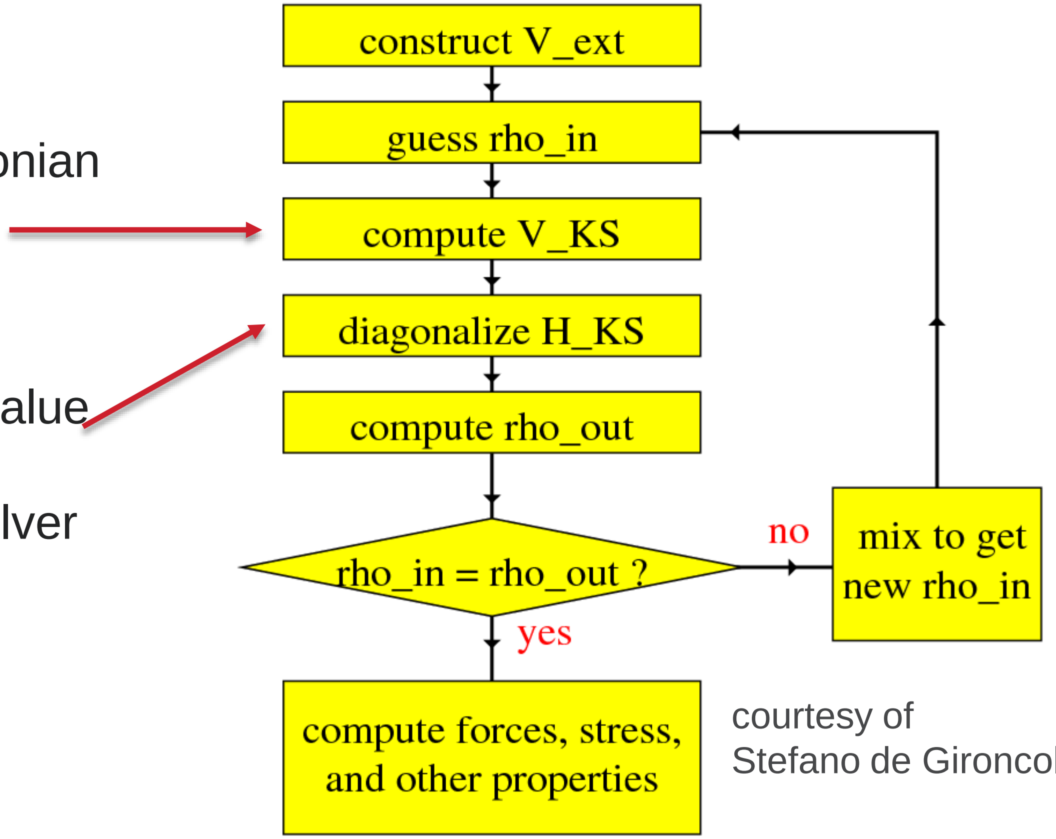


The above figure courtesy of P. Giannozzi and F. Spiga

MAJOR KERNELS

- Constructing subspace Hamiltonian (h_psi)
  - Two FFT for XC potential
  - 3D parallel FFT in FFTXlib
- Solving the generalized eigenvalue problem in subspace
  - Using Davidson iterative solver

Structure of a self-consistent type code



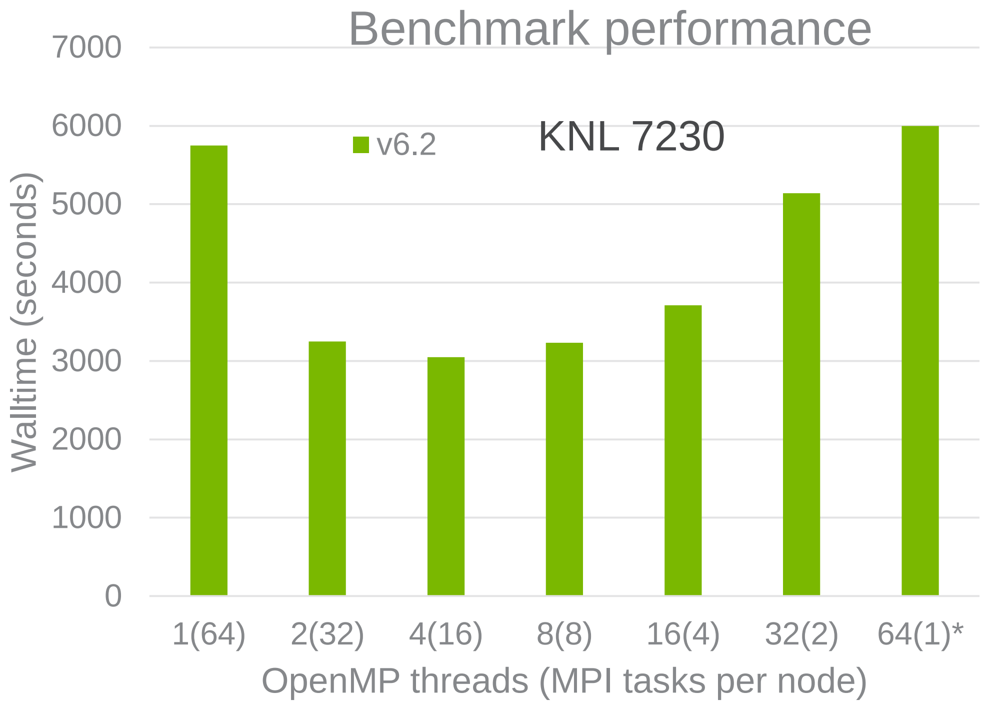
courtesy of
Stefano de Gironcoli

Argonne NATIONAL LABORATORY

THREADING IS NEEDED

v6.2.1 release performance is not satisfactory

- MPI only based on planewave decomposition hits a limit
  - Massive amount of small messages in parallel FFT
  - Saturated strong scaling performance
- Data not decomposed over MPI dominates the footprint
  - <psi|beta> for pseudopotentials >50%
- Better interoperability with accelerators



HOW TO IMPROVE THE THREADING

THE CURRENT STATUS

- What has already been threaded?
  - BLAS/ScaLAPACK especially ?gemm, part of the parallel eigensolver
  - 1D batched FFTs via DFTI
  - Limited places with explicit !$omp parallel do
- What is not threaded (shown by the Rice HPCToolkit trace)?
  - Case 1: Data transpose in FFTXlib, mostly memory operations
  - Case 2: Pseudopotential projectors
  - Case 3: Fortran array section pitfall
- What is the criteria of threading necessity?
  - Loop iteration increases as the number of MPI reduces.
  - Distributed planewaves and real-space grid points

CASE 1: DATA TRANSPOSE IN FFT

Collapse threaded outer loops and leave innermost loop for SIMD

```
offset = 0
DO iproc2 = 1, nproc2
  kdest = ( iproc2 - 1 ) * sendsize
  kfrom = offset
  DO k = 1, ncp_(me2)
     DO i = 1, desc%nr2p( iproc2 )
      f_aux ( kdest + i ) =  f_in ( kfrom + i )
    ENDDO
    kdest = kdest + nr2px
    kfrom = kfrom + desc%nr2x
  ENDDO
  offset = offset + desc%nr2p( iproc2 )
ENDDO
```

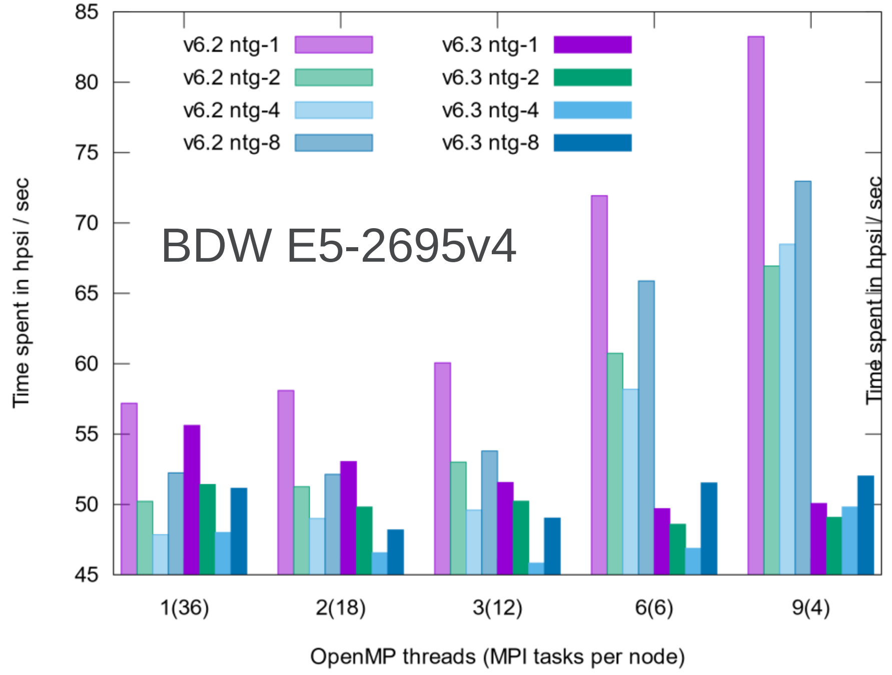
```
!$omp parallel do collapse(2) private(kdest,kfrom)
   DO iproc2 = 1, nproc2
     DO k = 0, nr1p_(me2)*desc%my_nr3p-1
       kdest = ( iproc2 - 1 ) * sendsize + nr2px * k
       kfrom = desc%nr2p_offset(iproc2)
               + desc%nr2x *k
       DO i = 1, desc%nr2p( iproc2 )
         f_aux ( kdest + i ) =  f_in ( kfrom + i )
       ENDDO
     ENDDO
   ENDDO
!$omp end parallel do
```

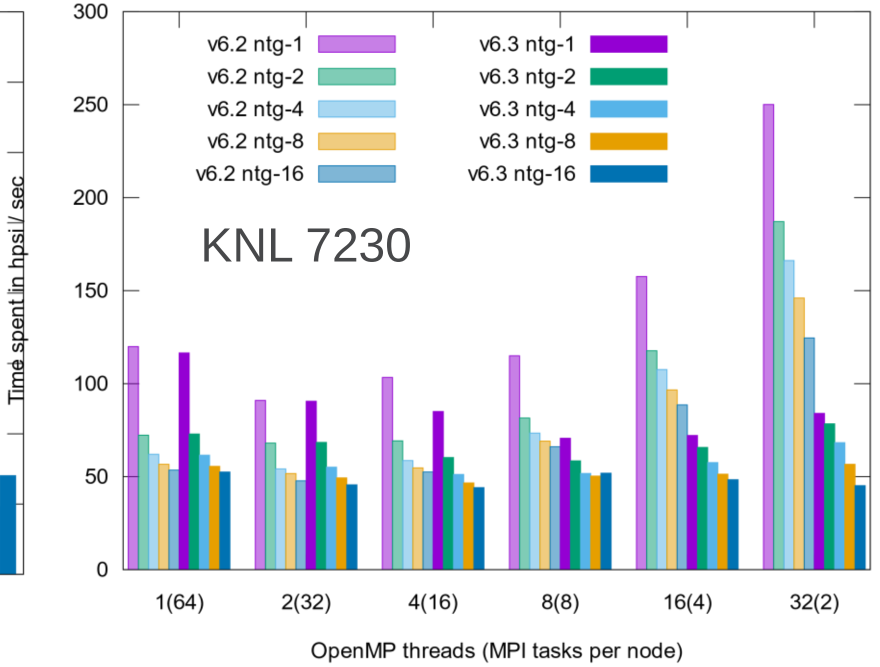
Precompute offsets

4 places in xy and yz transpose

TIME SPENT IN HPSI

Nearly constant in PSIWAT benchmark, on 32 nodes of Bebop





CASE 2: PSEUDOPOTENTIAL PROJECTORS

Chunked planewaves for threading, cache blocking and vectorization

```
DO nt = 1, ntyp
  DO na = 1, nat
    IF (ityp (na) == nt) THEN
      DO ih = 1, nh(nt)
        ikb = indv_ijkb0(na) + ih
        DO ig = 1, npw
          ar = vkb (ig, ikb)*conjg(vkb (ig, ikb))
          h_diag (ig) = h_diag (ig) + …
          s_diag (ig) = s_diag (ig) + …
        ENDDO
      ENDDO
    ENDIF
  ENDDO
ENDDO
```

```
!$omp parallel do private(…)
DO iblock = 1, numblock
  DO nt, na, ih
    ikb = indv_ijkb0(na) + ih
    DO ig = (iblock-1)*blocksize+1, MIN(iblock*blocksize, npw)
      ar = vkb (ig, ikb)*conjg(vkb (ig, ikb))
      h_diag (ig) = h_diag (ig) + …
      s_diag (ig) = s_diag (ig) + …
    ENDDO
  ENDDO
ENDDO
!$omp end parallel do
```

blocksize=256

CASE 3: FORTRAN ARRAY SECTION PITFALL

Simple to write but critical to performance

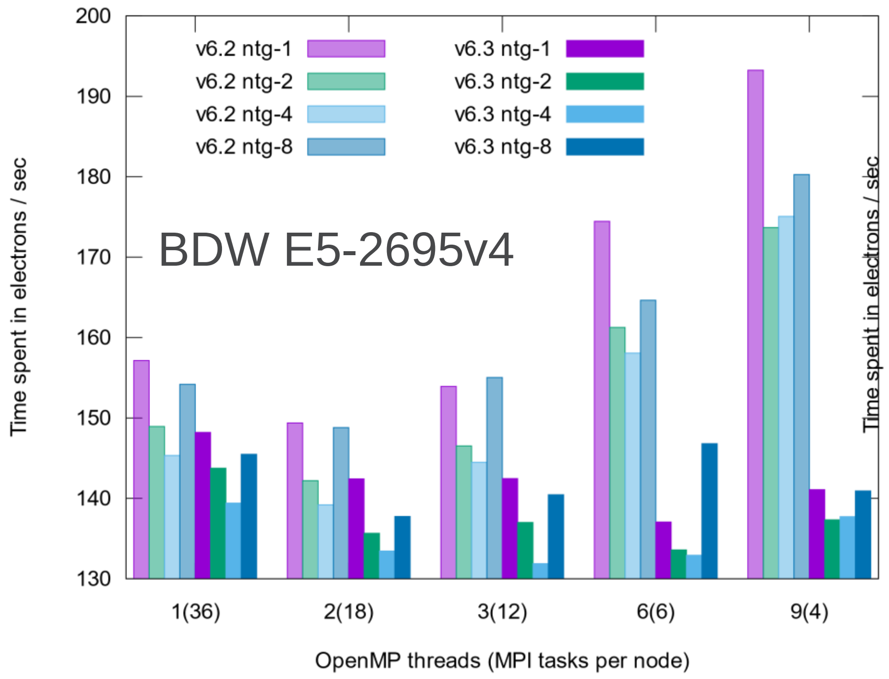
```
ALLOCATE( psi( npwx, npol, nvecx ) )
ALLOCATE( hpsi( npwx, npol, nvecx ) )
hpsi = (0.d0,0.d0)
psi  = (0.d0,0.d0)
psi(:,:,1:nvec) = evc(:,:,1:nvec)
```

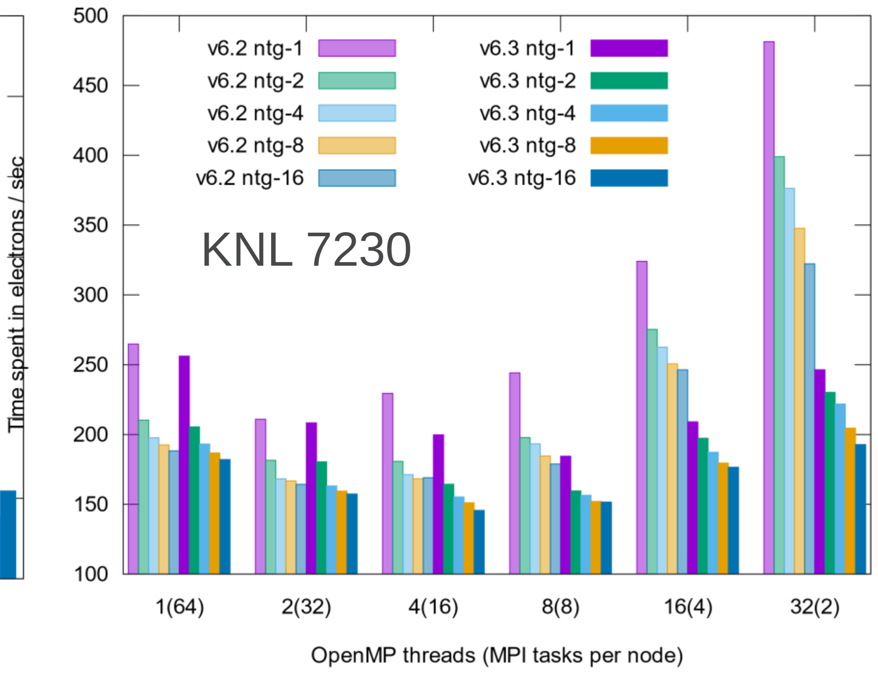
```
CALL threaded_memcpy(psi, evc,
                             nvec*npol*npwx*2)
```

- Array section can be expensive when the array is large.
- Array section is not threaded
- !$omp workshare is not threaded with Intel compiler
- Setting psi, hpsi to zero is a waste
- SMP full memory BW requires threading

TIME SPENT IN SCF

Significant improved performance at large thread counts

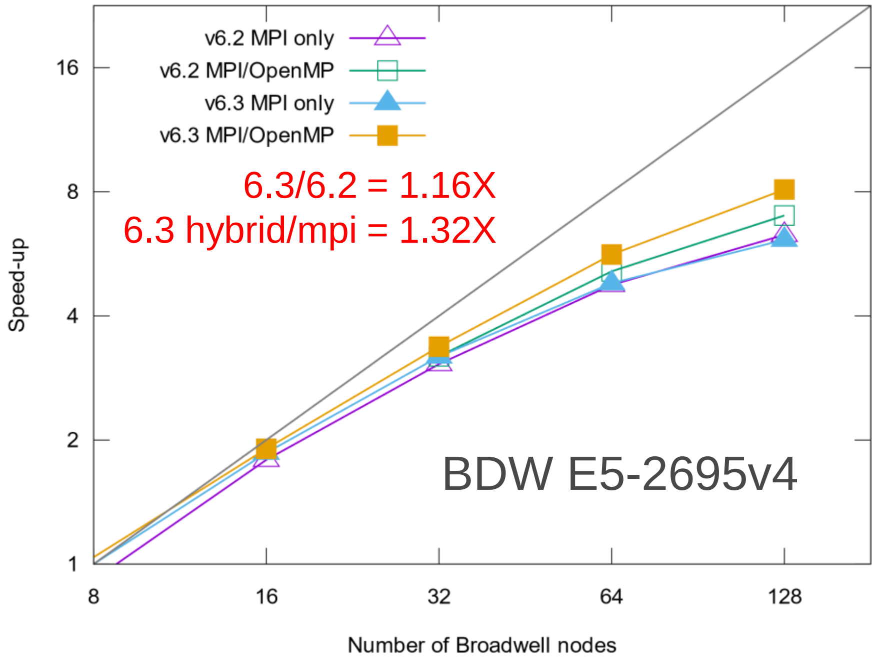


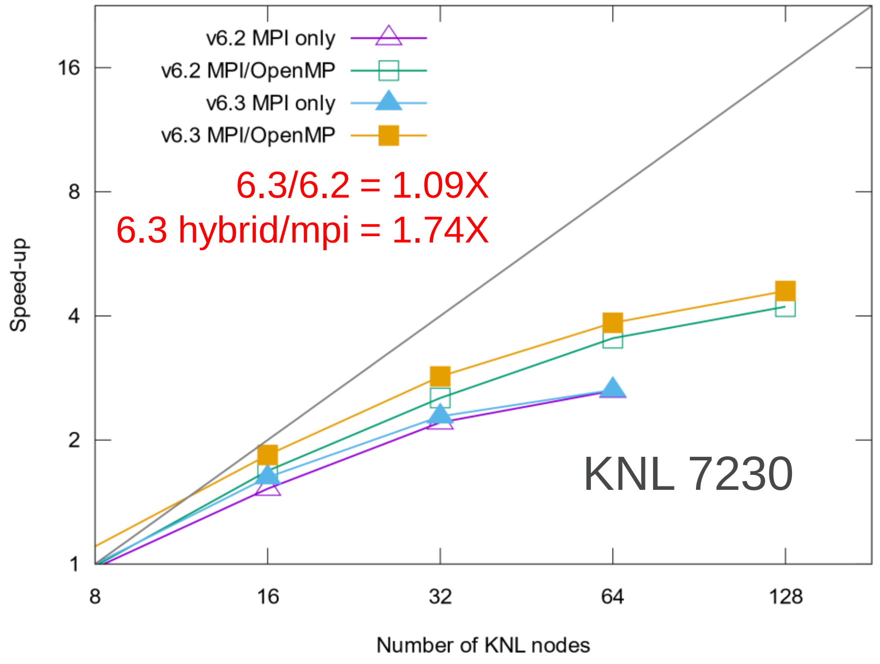


IMPROVED PERFORMANCE

STRONG SCALING

Using MPI/OpenMP is a must, particularly KNL





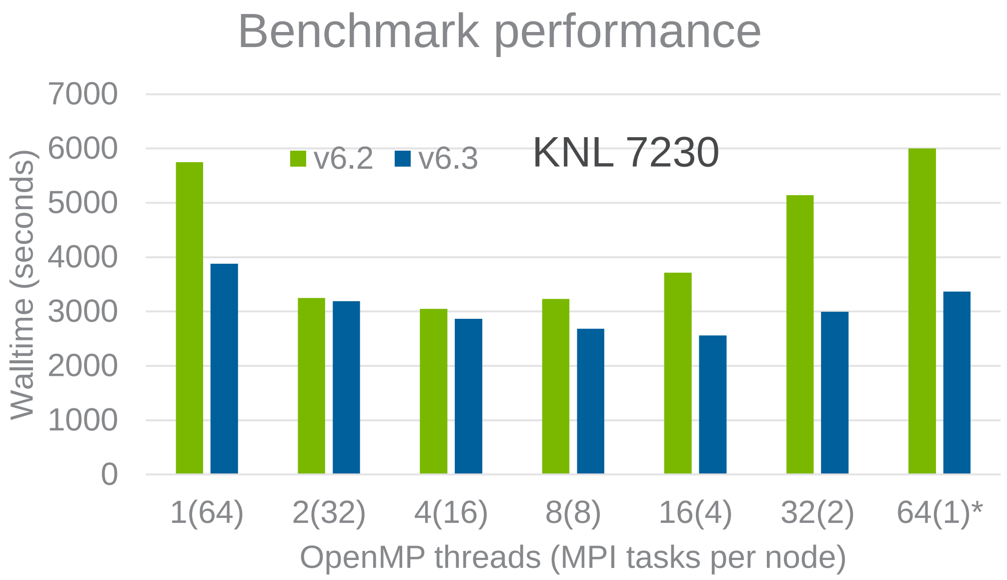
Argonne NATIONAL LABORATORY

TIME TO SOLUTION

Improved both MPI and MPI/OpenMP hybrid cases

- 662 atoms, 6103 electrons
- 45 Ry ecutwfc, 275 Ry ecut rho
- 6 k-point with pool parallelization
- 4 task groups
- 96 KNL 7230 nodes, no HT

- Good performance is no more limited to small thread counts.
- Start with 4 threads/MPI on KNL



Q&A