Parallel Performance optimisation in ONETEP

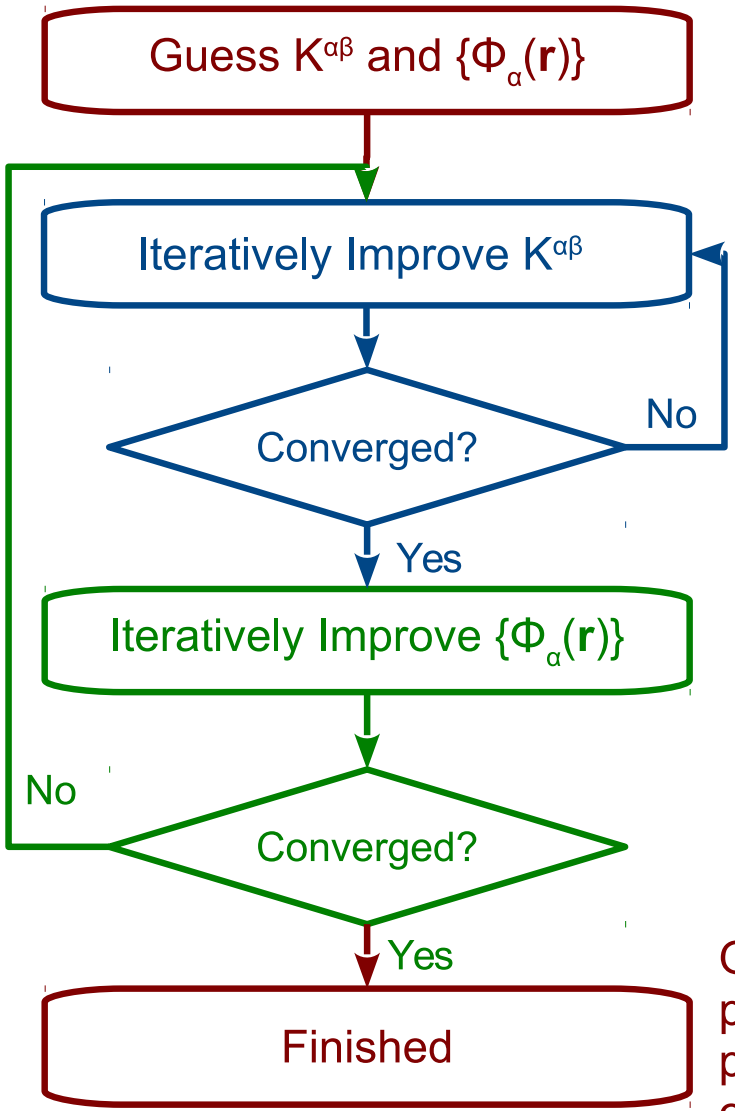
Nicholas D.M. Hine

Associate Professor, Theory Group, Physics, University of Warwick

ONETEP Masterclass 2017

ONETEP: Linear Scaling DFT

Initial guesses based on pseudoatomic orbitals for $\{\Phi_\alpha(\mathbf{r})\}$ and non-SC solution of $H_{\alpha\beta}$ for atomic densities



Outer loop: Optimise total energy wrt $\{\Phi_\alpha(\mathbf{r})\}$, minimising wrt $K^{\alpha\beta}$ at each step

Inner loop: Optimise total energy wrt $K^{\alpha\beta}$ for fixed $\{\Phi_\alpha(\mathbf{r})\}$ (constraints of normalisation and idempotency)

Output energy, forces, populations etc. Can post-process for eigenstates if req'd.

Computational Effort in ONETEP - local potential matrix elements

1. Communication of grid data (extract)

$$V(\mathbf{r}) \text{ (global)} \quad \rightarrow \quad V(\mathbf{r}) \text{ (local)}$$

2. FFTs (interpolate/filter) in FFTBox

$$V(\mathbf{r})\phi_\beta(\mathbf{r}) = \mathscr{F}\mathscr{F}\left[V(\mathbf{r}) \times \mathscr{F}\mathscr{I}\left[\phi_\beta(\mathbf{r})\right]\right]$$

3. NGWF comms & operations in FFTBox

$$\langle \phi_\alpha | \hat{V} | \phi_\beta \rangle = \int_{\mathrm{loc}(\alpha)} \phi_\alpha(\mathbf{r}) V(\mathbf{r}) \phi_\beta(\mathbf{r}) \mathrm{d}\mathbf{r}$$

4. Sparse matrix algebra

$$(KH)^\alpha{}_\beta = K^{\alpha\gamma} H_{\gamma\beta}$$

Many other parts (sparse matrix setup, Ewald, whole-cell FFT, etc) but these are usually comparatively trivial compared to those above

Computational Effort in ONETEP - density

1. Sparse matrix algebra

$$(KS)^{\alpha}{}_{\beta} = K^{\alpha\gamma} S_{\gamma\beta}$$

2. NGWF comms & operations in FFTBox

$$\left[\sum_{\beta} K^{\alpha\beta}\phi_{\beta}(\mathbf{r})\right]_{\text{coarse grid}}$$

3. FFTs (interpolate/filter) in FFTBox

$$\rho_{\alpha}(\mathbf{r}) = \mathscr{F}\mathscr{I}\left[\phi_{\alpha}(\mathbf{r})\right] \times \mathscr{F}\mathscr{I}\left[\sum_{\beta} K^{\alpha\beta}\phi_{\beta}(\mathbf{r})\right]$$

4. Communication of grid data (deposit)

$$\rho(\mathbf{r})\ (\text{global}) = \sum_{\alpha} \rho_{\alpha}(\mathbf{r})\ (\text{local})$$

Many other parts (sparse matrix setup, Ewald, whole-cell FFT, etc) but these are usually comparatively trivial compared to those above

Main Data Structures

- FFT boxes: stored for a 'batch' of NGWFs simultaneously
- Whole cell grids (3-10 stored at any one time, dependent on options), parallelised over slabs in 12-direction (real space)
- Sparse Matrices (SPAM3 type) parallelised over columns
- Workspaces (300-500MB, depending on options)

Grouped Communications: nodes share data. Default group size is closest power of two to square-root of number of processes (can adjust with comms_group_size)

All-MPI Parallelism model has high memory requirements for high-accuracy runs: often >2GB/core
Also, MPI Parallelism reaches saturation much below around 10 atoms per core for large jobs

Hybrid Parallelism

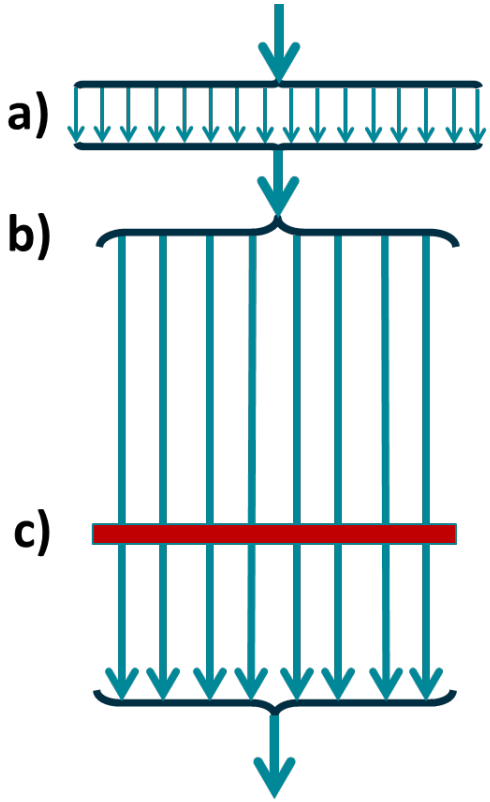
MPI Parallelism

- Message Passing Interface
- Splits code into lots of MPI 'processes', each running the same code
- Performance dependent on interconnect speed between nodes
- Uses shared memory for messages between processes on same node, but still copies between memory locations

OpenMP Parallelism

- Open Multi-Processing
- Shared-Memory multithreaded model - direct access by one thread to memory of another
- Runs one 'master' thread, which splits into multiple threads inside PARALLEL regions
- Only acts within a node

OpenMP Paradigm

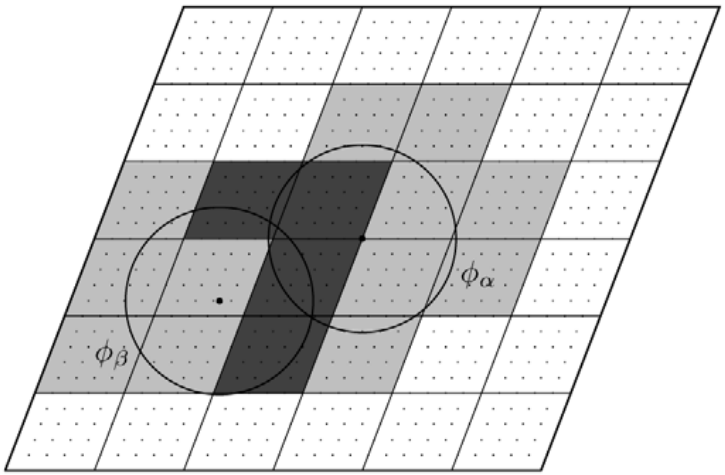



```
!$OMP PARALLEL NUM_THREADS(16)

!$OMP END PARALLEL

!$OMP PARALLEL NUM_THREADS(8)
!$OMP DO
   do i = 1, numit
      !Parallel code

!$OMP CRITICAL
      !Serially executed code
!$OMP END CRITICAL

   end do
!$OMP END PARALLEL
```

K. A. Wilkinson, N. D. M. Hine, and C.-K. Skylaris Hybrid MPI-OpenMP parallelism in the ONETEP linear-scaling electronic structure code: Application to the delamination of cellulose nano-fibrils J. Chem. Theory Comput. 10, 4782(2014)

Computational Effort in ONETEP

1. FFT Row Sum operations



Communicate NGWF data, deposit to large arrays

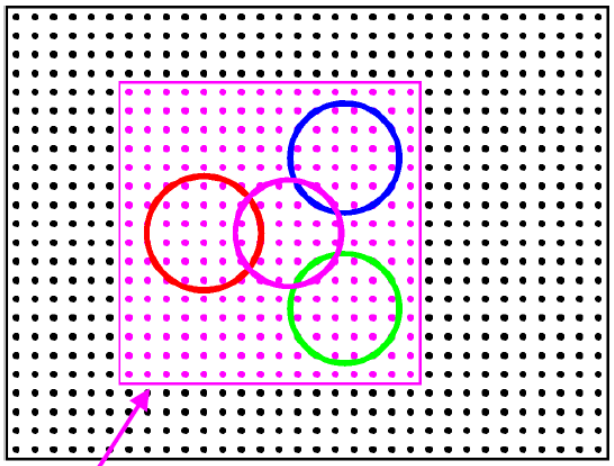
$$n(\mathbf{r}) = \sum_{\alpha\beta} \phi_\alpha(\mathbf{r}) K^{\alpha\beta} \phi_\beta(\mathbf{r})$$

$$= \sum_{\alpha} \phi_\alpha(\mathbf{r}) \left[\sum_{\beta} K^{\alpha\beta} \phi_\beta(\mathbf{r}) \right]_{\text{FFTbox}}$$

- First attempt: thread-parallelise deposition of each $\phi_\beta(\mathbf{r})$ to FFTbox $\left[\sum_\beta K^{\alpha\beta}\phi_\beta(\mathbf{r})\right]_{\text{FFTbox}}$ by PPDs
  - Not much speedup, sometimes slowdown from trying to write to nearby cache lines of box
- Second attempt: for each $\phi_\beta(\mathbf{r})$ communicated, thread-parallelise over boxes to deposit it to
  - Much better: near-linear speedup with thread count within deposition section.
  - Can be a source of performance loss if comms is slow (need cleverer buffering)

Computational Effort in ONETEP

2. FFT box Fourier Transforms



C.-K. Skylaris, A. A. Mostofi, P. D. Haynes, C. J. Pickard & M. C. Payne, *Comp. Phys. Comm.* **140**, 315 (2001)
A. A. Mostofi, C.-K. Skylaris, P. D. Haynes & M. C. Payne, *Comp. Phys. Comm.* **147**, 788 (2002)

3D FFTs on quite large box of local data: major part of density, local potential, nonlocal projectors, NGWF gradient operations

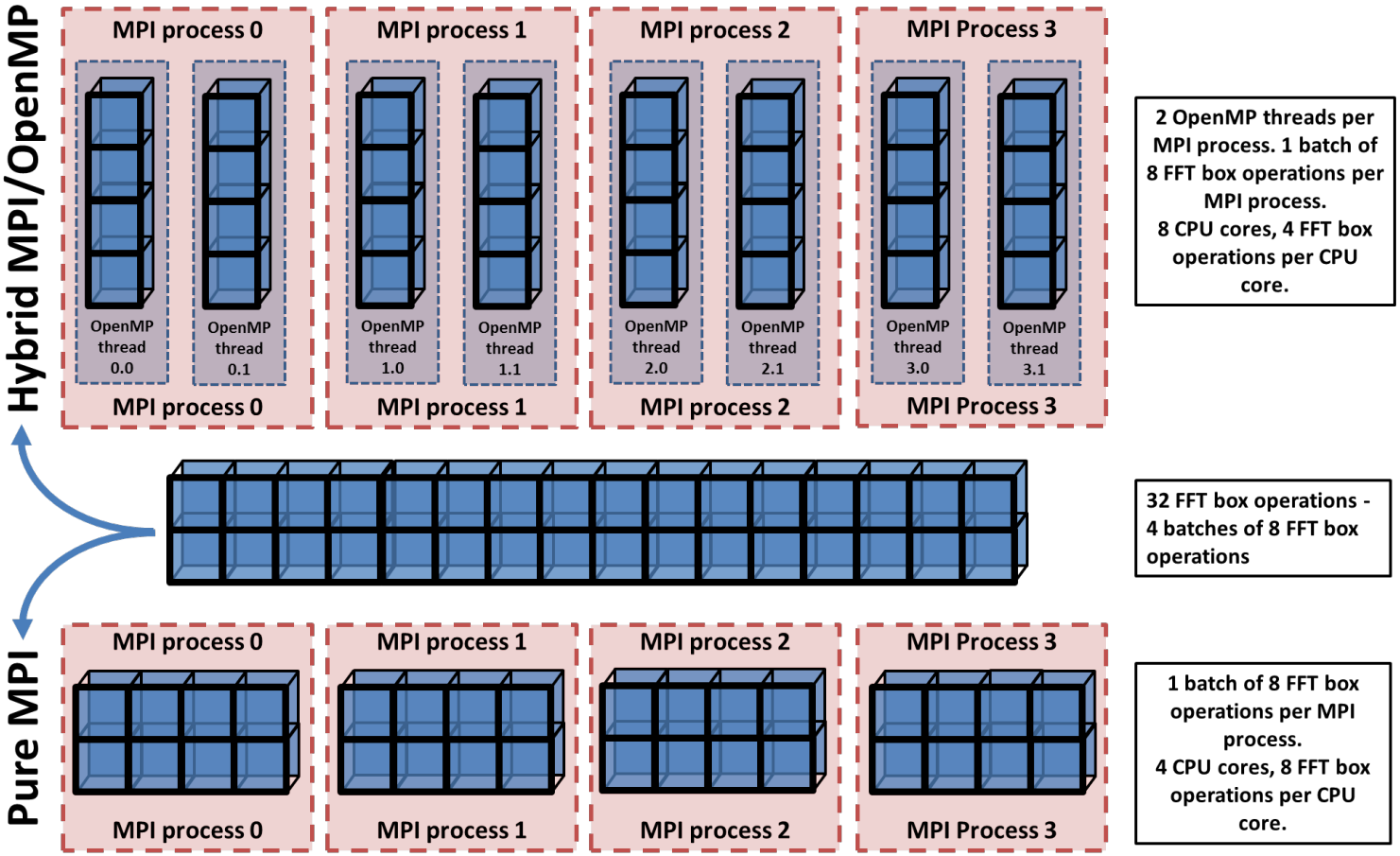
FFT-box routines exhibit perfect scaling due to complete locality of data.
Reminder: density operation involves interpolation to fine grid due to product of two NGWFs:

$$\rho(\mathbf{r}) = \sum_{\alpha\beta} \phi_\alpha(\mathbf{r}) K^{\alpha\beta} \phi_\beta(\mathbf{r})$$

- Parallelise over interpolate/filter operations on each FFTbox of a given column function $\phi_\alpha(\mathbf{r})$
- Needs fine/coarse workspace arrays for each core (large)
- Typical Size: $(150\text{-}250)^3$ (50-250 MB)

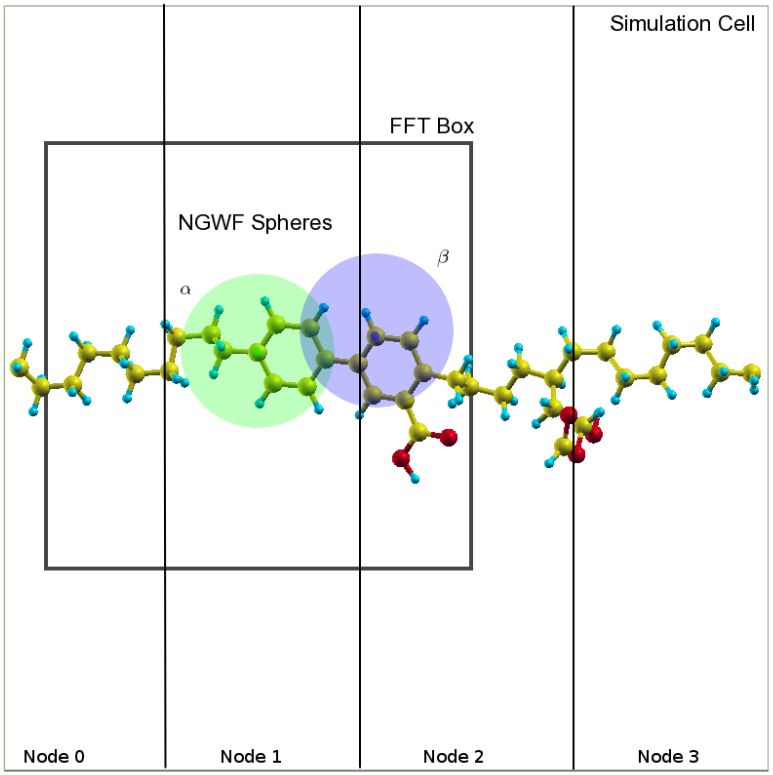
Computational Effort in ONETEP

2. FFT box Fourier Transforms



Computational Effort in ONETEP

3. Whole Cell Grid Extract/Deposit

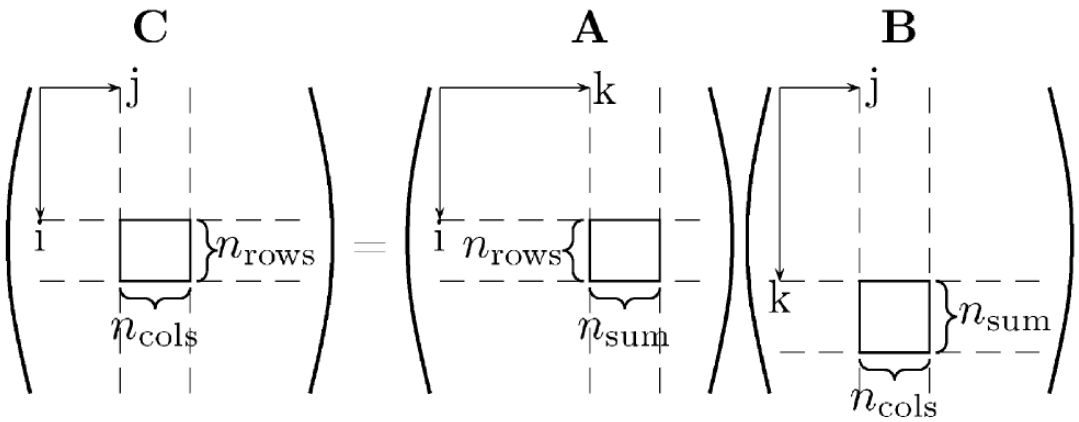


Transfer of local boxes to/from distributed whole-cell arrays (deposit density, extract potential)

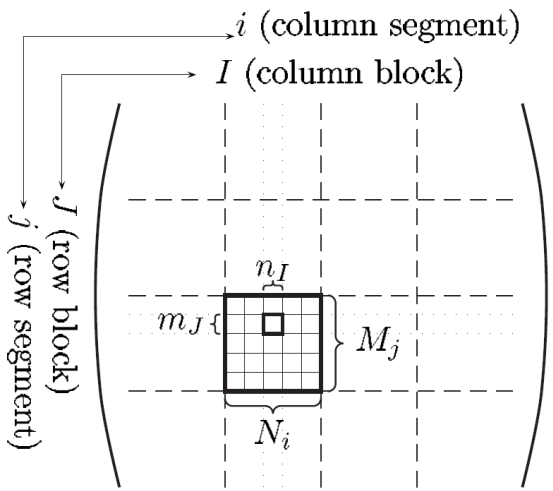
- Currently not thread parallel: occurs inside `!$OMP CRITICAL` regions so execution is limited to one MPI process at a time
- Ideally, would funnel all comms through root process and overlap comms with compute (hard to structure)
- Also scope to reduce total amount of comms by accumulating density in a big box spanning all the FFTboxes in a batch
- Or by communicating box limits for a batch in advance and working out when comms can be skipped entirely (avoids synchronisation)

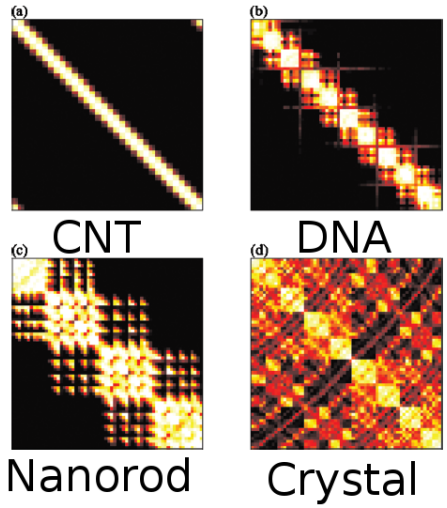
Computational Effort in ONETEP

4. Sparse Matrix Algebra



Communicates matrix data, multiplies segments.





Computational Effort in ONETEP

4. Sparse Matrix Algebra

Thread-parallelise segment-segment pair operations:

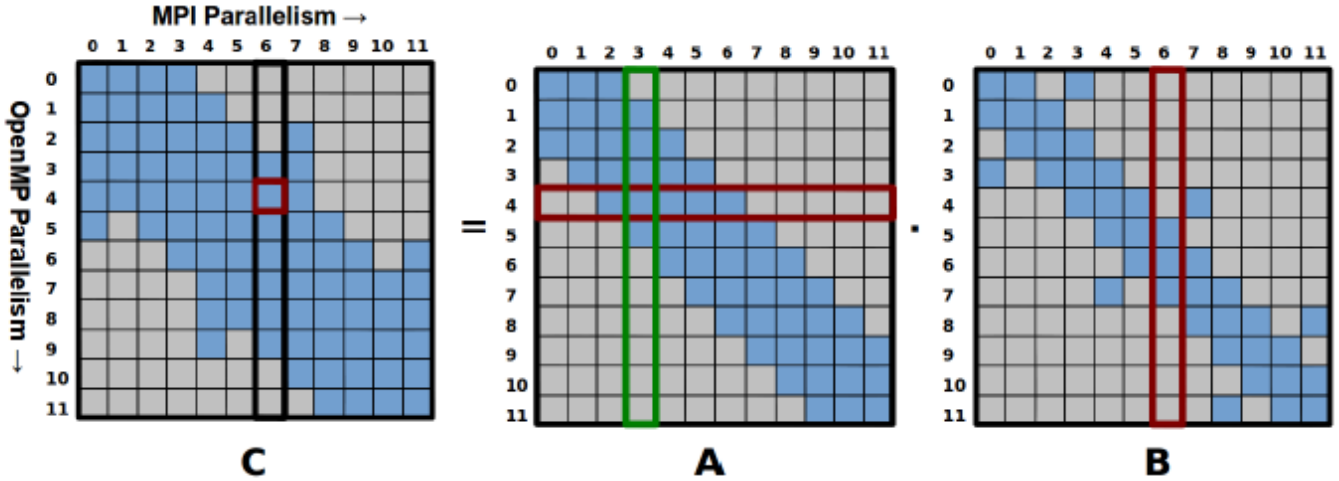


Figure 3: Schematic of the parallel decomposition of the workload for a sparse matrix multiplication under the hybrid OpenMP-MPI scheme, on 12 MPI processes. Blue shading indicates segments containing non-zero elements. The red boxes highlight a specific segment of **C** local to MPI process 6 and the range of segments of **A** and **B** which contribute to it. The green box indicates the set of segments communicated by MPI process 3, of which only some are non-zero. The OpenMP parallelism divides up the workload of each MPI process by dynamically distributing the segment pair matrix product operations between available threads.

- Requires $n_{threads} < n_{processes}$ to get any speedup.
- Alternative would be to multi-thread the DGEMM calls but this is not generally very efficient except for huge matrices

OpenMP thread controls

Controlled by four input variables or environment variables:

- Thread count in routines with minimal workspace (Ewald, Sparse Algebra..):
  - `threads_max` (pub_threads_max internally)
  - defaults to OMP_NUM_THREADS
- Thread count in FFTBox-based routines (density, locpot, kinetic, projectors)
  - `threads_num_fftboxes` (`pub_threads_num_fftboxes` internally)
  - defaults to OMP_NUM_THREADS
- Thread count WITHIN EACH BOX FFT
  - `threads_per_fftbox` (pub_threads_max internally)
  - defaults to 1 as not always supported (depends on libraries)
- Thread count in whole-cell FFTs (hartree, GGAs, van der Waals DF, etc)
  - `threads_per_cellfft` (pub_threads_max internally)
  - defaults to 1 as not always supported (depends on libraries)

Stack size considerations

ONETEP requires a reasonably large 'stack' available: some clusters set this to be very small by default

- If you get a crash right at the start, ensure your script runs this command before launching the code
  - `ulimit -s unlimited`
- However, this only affects the Master thread. Other thread stacks controlled by environment variable
  - OMP_STACKSIZE = 64M
- Running with the intel compiler version 17, and Intel MPI 17, you may experience an issue to do with interoperability between the two which leads to data corruption. If you get nonsense, try:
  - export I_MPI_OFA_TRANSLATION_CACHE=0

Memory Usage

- FFT boxes: number controlled by `fftbox_batch_size`
- Use at least threads_num_fftboxes * 2 * n, n=1 is fine, sometimes higher is better.:
  - Less repeated NGWF comms (ideal is only send each NGWF once)
  - But they take a lot of memory! 16×64MB=1GB per MPI proc
- Whole cell grids: keep an eye on how much time is spent in these operations as it does not scale down with thread count
- Memory of workspaces scales up with number of fftbox threads (300-500MB / proc, depending on options)

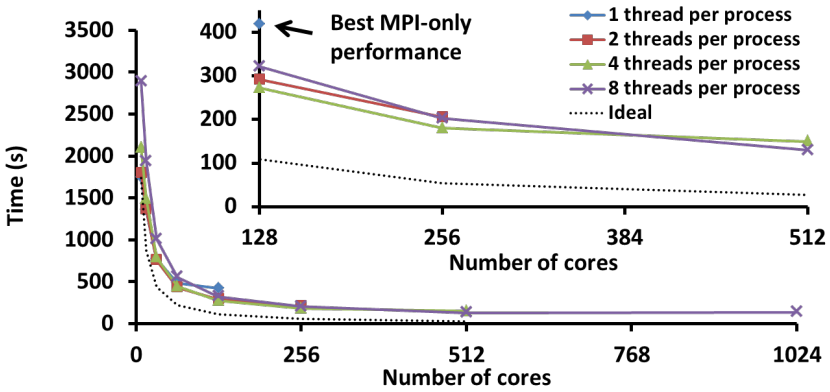
Grouped Communications: nodes share data. Default group size is closest power of two to square-root of number of processes.

- Sparse matrix operations: timings have considerable dependence on comms_group_size

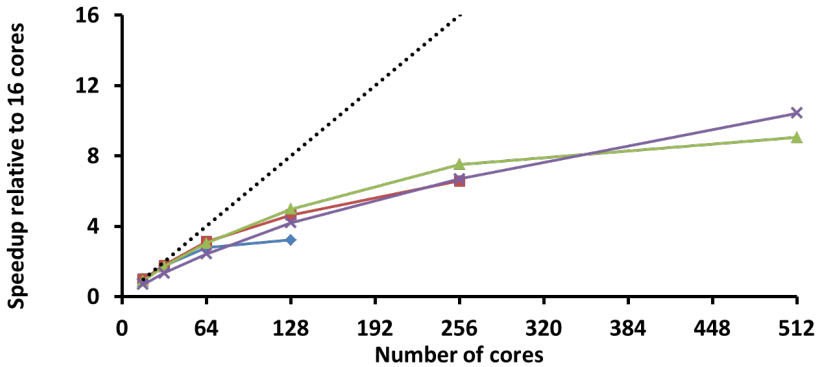
All-MPI Parallelism needs more memory: 2GB / proc minimum
With OpenMP, can go down to 1GB / proc or below

Performance in Small Systems

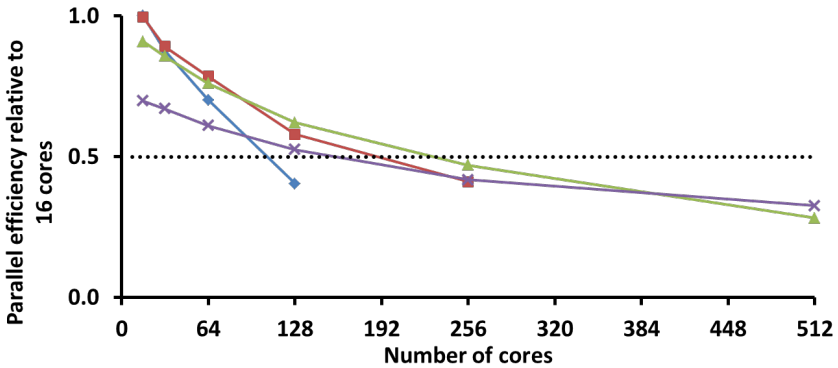
a)



b)



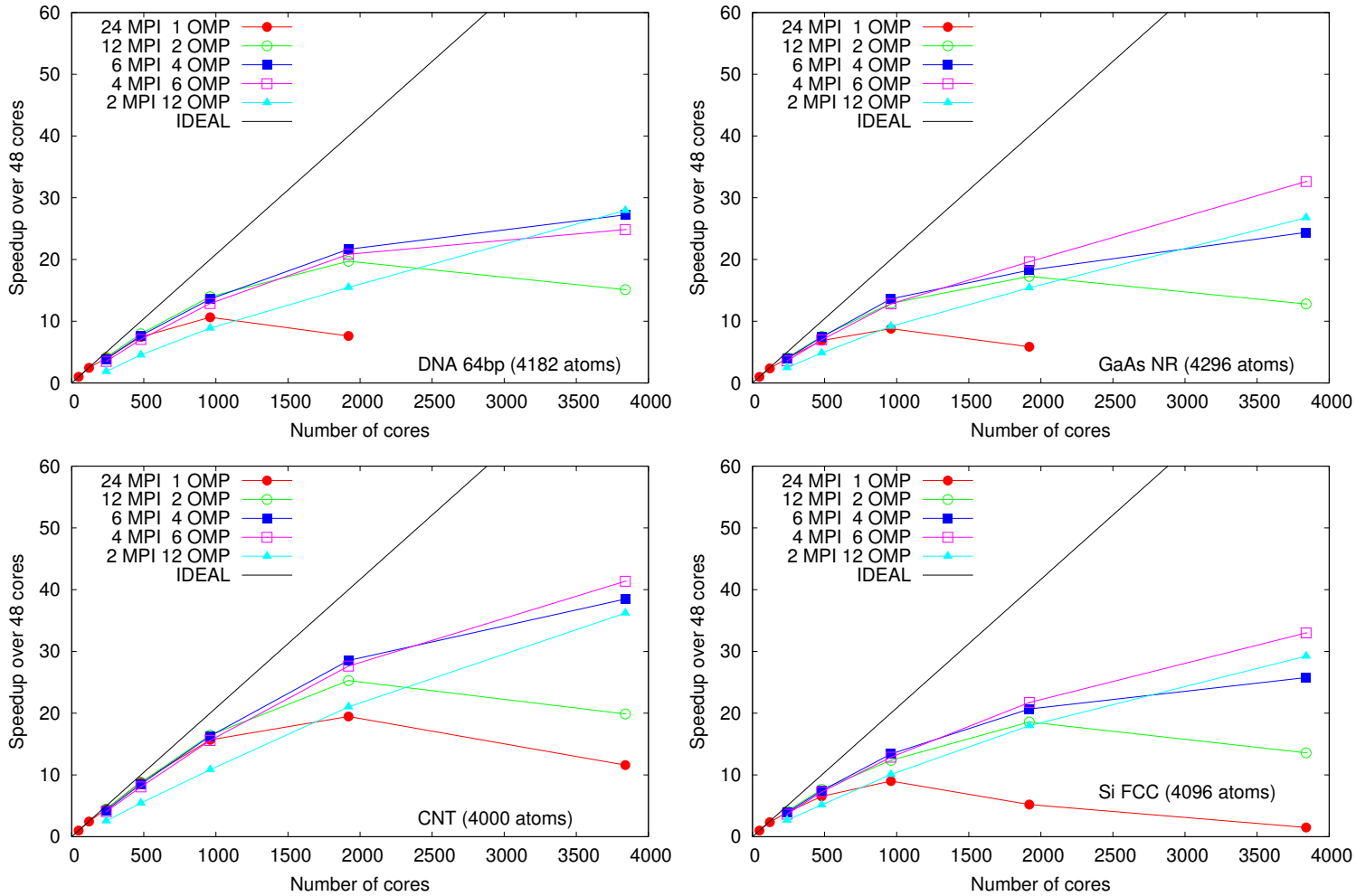
c)



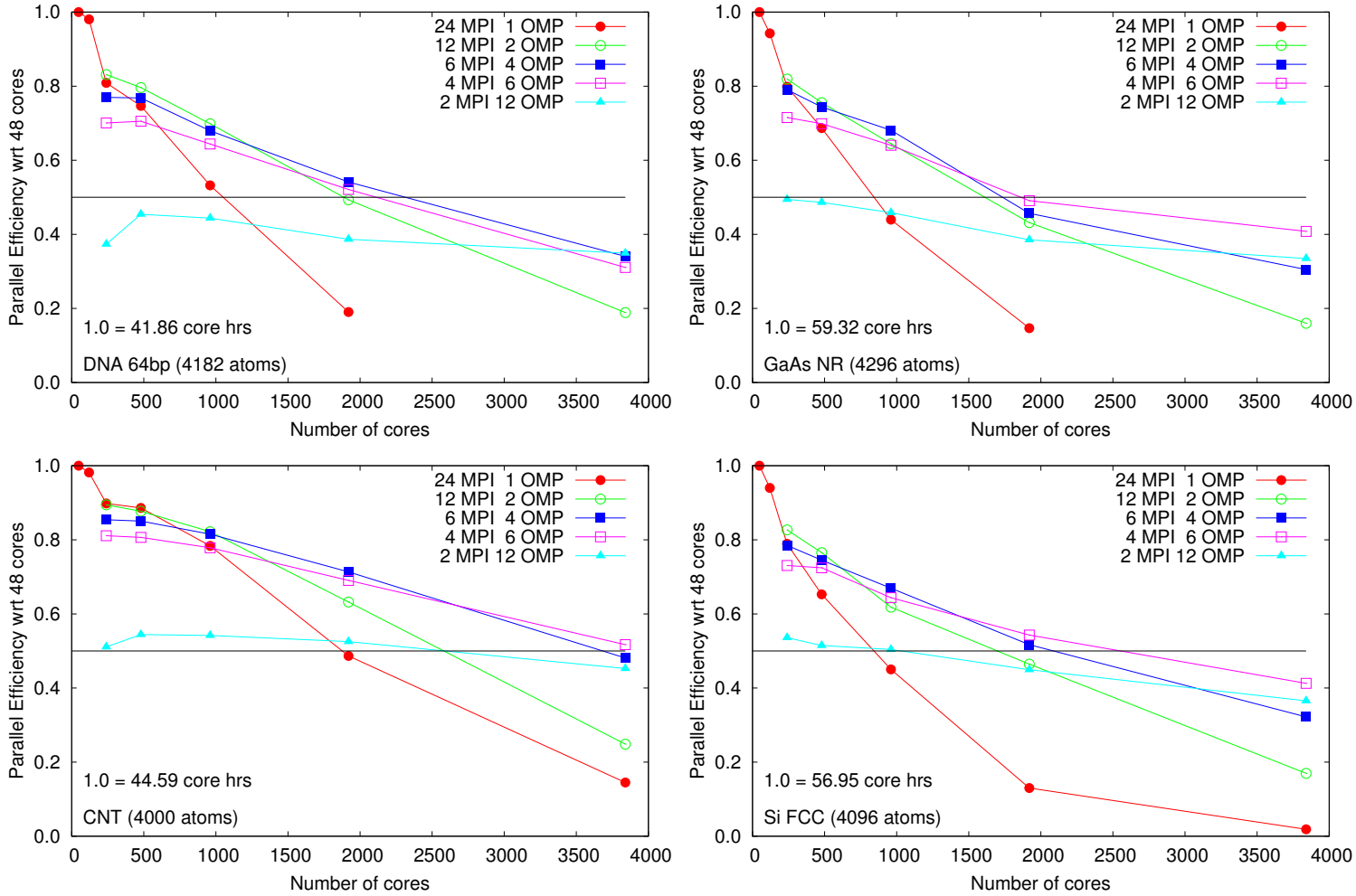
189 atom nucleotide sequency in vacuum.
Calculations performed on a using the Iridis 4 supercomputer (Southampton)

Large Systems - ARCHER tests

Currently, each thread comes with a parallel efficiency hit due to routines with no OpenMP and MPI-collectives



Large Systems - ARCHER tests



On more realistic systems ~4000 atoms: dropoff in parallel efficiency due to limitations of sparse algebra and whole-cell grid ops. Still scales well to ~2000 cores.

Very Large Systems - BG/Q tests

Amyloid Fibril: production-quality settings (8a0 NGWFs, 800eV psinc grid, 40a0 kernel cutoff), 13696 atoms, 36352 NGWFs. Excellent scaling to 16384 cores.

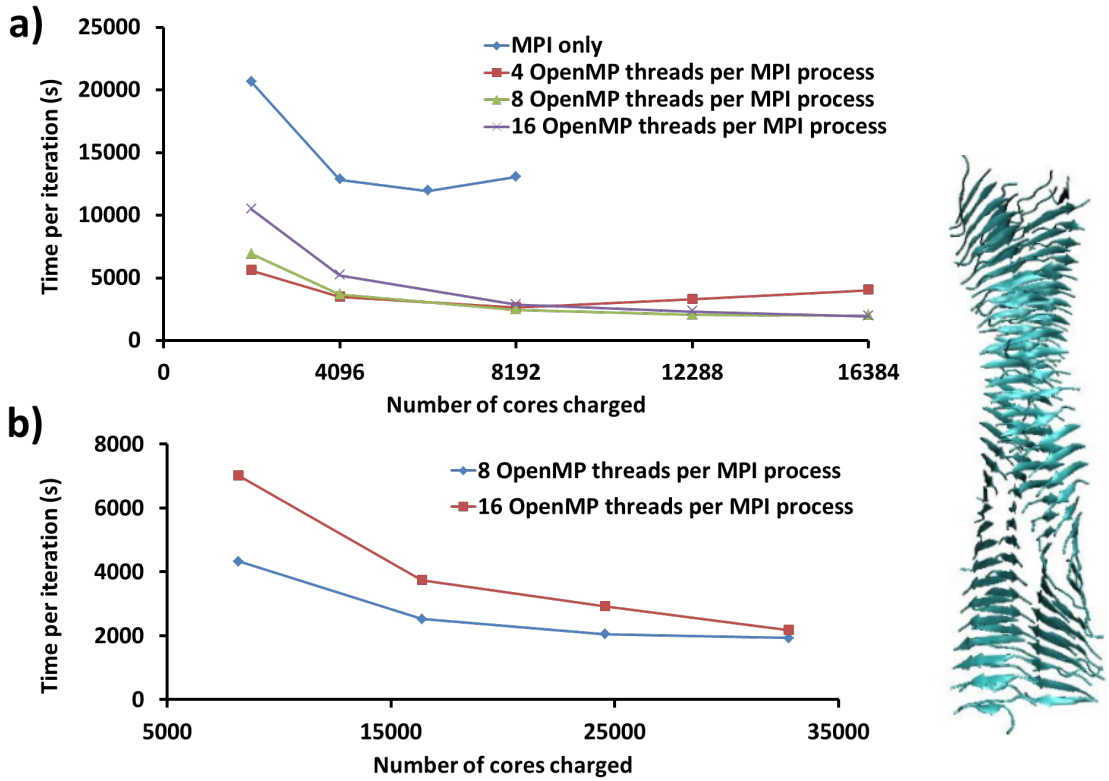


Figure 8: a) Total time for 1 iteration of the 13,969 atom beta-amyloid fibril, for MPI-only (blue), and 4/16, 8/32 and 16/64 OpenMP threads per MPI process (red, green, purple respectively). b) Total time for the 41,907 atom Amyloid fibril trimer. Both sets of calculations consisted of 1 iteration of the NGWF optimisation loop with production-quality settings (5 iterations of the density kernel loop, 30.0 a_0 density kernel cutoff, 8.0 a_0 NGWF radii, 800 eV psinc kinetic energy cutoff)

Timings output

If you are concerned your simulations are not achieving good parallel efficiency, feel free to check with the developers

- Set `timings_level = 3` to get "self-timings" (i.e. time spent in that routine and nowhere else)
- If you want to compare timings between runs, set timings_order = NONE to leave timer output un-sorted.

```
==================== AVERAGE TIMINGS FROM ALL NODES (SELF) ====================
++ TAG                                  #calls    cpu time   %total  Gflops ++
++ basis_extract_function_from_box          60       0.00s   0.001%  ------ ++
++ hartree_on_grid                           9       0.01s   0.001%  ------ ++
++ paw_tcore_density                         1       0.01s   0.002%  ------ ++
++ paw_tcore_hartree_on_grid                 1       0.01s   0.002%  ------ ++
++ aug_nonlocal_mat                          9       0.03s   0.005%  ------ ++
++ ngwfs_initialise_from_radial              1       0.04s   0.006%  ------ ++
++ sparse_hotelling_invert                   3       0.04s   0.006%  ------ ++
++ ngwf_gradient_coeffs                      1       0.05s   0.007%  ------ ++
++ sw_init                                  19       0.06s   0.008%  ------ ++
++ integrals_locpot_dbl_grid                 7       0.12s   0.017%  ------ ++
++ kernel_fix                               4       0.23s   0.034%  ------ ++
++ hamiltonian_energy_components            1       0.39s   0.057%  ------ ++
++ fourier_interpolate_cell                 8       0.49s   0.070% 319.958 ++
++ ewald_calculate_energy                   1       0.53s   0.076%  ------ ++
++ kernel_purify                           12       0.53s   0.077%  ------ ++
++ basis_copy_function_to_box              575       0.59s   0.085%  ------ ++
++ paw_dataset_init                         3       0.67s   0.096%  ------ ++
++ paw_dij_xc                              10       0.84s   0.121%  ------ ++
++ lnv_gradient_norm                        4       0.87s   0.125%  ------ ++
++ density_init_guess_real                  1       1.00s   0.144%  ------ ++
++ lnv_denskernel_optimise_cg               2       1.00s   0.144%  ------ ++
++ fourier_filter_cell                      8       1.03s   0.149% 151.743 ++
++ palser_mano_kernel_optimise              1       1.19s   0.171%  ------ ++
++ hamiltonian_lhxc_calculate               7       1.36s   0.196%  ------ ++
++ fourier_apply_box_pair                 106       1.65s   0.237%  5.411k ++
++ restart_kernel_write                     3       2.09s   0.301%  ------ ++
++ nlxc_vdw_energy                         18       2.17s   0.312%  ------ ++
++ pseudo_make_structure_factor             1       2.36s   0.340%  ------ ++
++ xc_energy_potential                      9       2.48s   0.356%  ------ ++
++ density_on_dbl_grid                      8       3.56s   0.512%  ------ ++
++ sparse_init                              1       4.44s   0.639%  ------ ++
++ paw_species_calc_proj_prec_mat           1       4.50s   0.647%  ------ ++
++ projectors_init_fftbox_recip            10       4.63s   0.667%  ------ ++
++ augmentation_screen_dij                 11       4.70s   0.677%  ------ ++
++ main_program_(onetep.F90)                1       4.87s   0.700%  ------ ++
++ restart_ngwfs_tightbox_output            1       5.33s   0.766%  ------ ++
++ function_ops_brappd_ketppd               3       5.61s   0.806%  ------ ++
++ sparse_trace                           263       5.93s   0.854%  ------ ++
++ cell_grid_extract_box                  375       6.22s   0.895%  ------ ++
++ density_fftbox_deposit_to_cell         280       6.36s   0.915%  ------ ++
++ projectors_func_ovlp_box                3       6.72s   0.966%  ------ ++
++ fourier_apply_cell_forward             261       7.34s   1.056% 626.390 ++
++ fourier_apply_cell_backward            280       7.63s   1.098% 646.461 ++
++ cell_grid_deposit_box                  552       8.29s   1.192%  ------ ++
++ augmentation_density_on_grid             8       9.53s   1.371%  ------ ++
++ density_fftbox_interpolate_multiply    280      10.63s   1.529%  ------ ++
++ integrals_kinetic                        3      13.90s   2.000%  ------ ++
++ basis_dot_function_with_box          540.7k      16.59s   2.387%  ------ ++
++ ngwf_gradient_batch                      3      21.46s   3.088%  ------ ++
++ basis_add_function_to_box            674.1k      23.42s   3.369%  ------ ++
++ ngwf_cg_optimise                         1      28.74s   4.135%  ------ ++
++ density_batch_interp_deposit            24      29.08s   4.183%  ------ ++
++ density_batch_row_sums                  24      30.49s   4.387%  ------ ++
++ ngwf_gradient_lnv                        1      30.85s   4.438%  ------ ++
++ potential_apply_to_ngwf_batch           21      32.37s   4.657%  ------ ++
++ integrals_locpot_mat_els_batch          21      33.63s   4.838%  ------ ++
++ sparse_product                         410      48.00s   6.905%  ------ ++
++ fourier_apply_box                     1645      57.78s   8.312%  8.119k ++
++ projectors_gradient_batch               3      84.28s  12.125%  ------ ++
++ projectors_grad_precond_batch           3     118.42s  17.036%  ------ ++
```

Conclusions

- Hybrid OpenMP/MPI parallelism extends strong scaling considerably
- Required OpenMP-parallelised loops at high levels, paying attention to load balance and avoid or hide CRITICAL regions
- General advice for most modern clusters: around 4 MPI processes per node, 4-8 threads per process
- Try to enable MPI process "pinning" with correct placement (check advice in cluster documentation for how to achieve this)

Reference:
K. A. Wilkinson, N. D. M. Hine, and C.-K. Skylaris Hybrid MPI-OpenMP parallelism in the ONETEP linear-scaling electronic structure code: Application to the delamination of cellulose nano-fibrils
J. Chem. Theory Comput. 10, 4782 (2014)

Acknowledgements

- ONETEP Developers Group: N. Hine, J. Dziedzic, A. Mostofi, C. Skylaris, P. Haynes, M. Payne
- Collaborators: K. Wilkinson (OpenMP, Soton), J. M. Escartin (Cambridge).
- Funding: EPSRC, The Leverhulme Trust (ECF), Winton Programme for the Physics of Sustainability
- Computing: UK National Supercomputers (HECToR, ARCHER), Imperial HPC Facility (CX1/CX2), Cambridge HPC Facility (DARWIN), STFC Hartree Centre (BlueJoule), Argonne National Laboratory (MIRA Supercomputer)