EPW performance plan report

Document Information

Reference Number	POP_PP_06 (EPW)
Author	Brian Wylie (JSC)
Contributor(s)	Ilya Zhukov (JSC)
Date	May 12, 2017

Notices: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.*

Contents

1 Background

Applicants Name: Samuel Poncé
Institution: University of Oxford, UK
Application Name: EPW, version 4.0.0
Programming Language: Fortran90
Programming Model: MPI
Source Code Available: yes (GPL)
Input data: GaN/epw-CB-4q (polar wurtzite gallium nitride crystal with 64 k-points) uniform fine grid
Performance study: performance plan following initial audit POP_AR_28
User description: Currently the EPW code relies on MPI parallelization and scales correctly up to 200 cores. We would like to improve scalability to 1000 cores and also optimize the code for improved performance. We would be happy to be have an audit to identify the bottlenecks in the code and focus on those.
Application Description: EPW (www.epw.org) is an Electron-Phonon Wannier code which calculates properties related to the electron-phonon interaction using Density Functional Perturbation Theory and Maximally Localized Wannier Functions. It is distributed as part of the Quantum ESPRESSO suite.
Testcase Description: 216 MPI processes on 9 compute nodes.
Machine Description: ARCHER Cray XC30 at EPCC, comprising 4920 compute nodes, with dual 12-core Intel Xeon E5-2697v2 (Ivy Bridge) 2.7 GHz processors sharing 64GB or memory and joined by two QPI links, connected via proprietary Cray Aries interconnect (Dragonfly topology). PrgEnv-intel using Intel 15.0.2.164 compilers.
Analysis tools: Score-P/2.0.2, Scalasca/2.3.1. Score-P default (compiler+MPI) instrumentation, combined with runtime measurement filter specifically for FFTXlib fftw routines.

2 Initial analysis (v0)

Two measurements were provided for initial analysis from executions with 216 MPI ranks (on 9 compute nodes): a version exploiting memory-saving 'etfmem' (which does more file I/O) as well as a default configuration. In contrast to the previously audited executions, a finer uniform grid (rather than a coarse random grid) was used, and simulation exploited restart files to focus on the `ephwann` phase which interpolates from real-space Wannier to a dense Bloch grid.

- There seems no significant difference in performance between the memory-saving 'etfmem' (39623 seconds) and 'default' execution (38923 seconds) configurations (Figure 1). The difference of less than 2% (also compared to uninstrumented reference executions) is most likely due to file I/O variations (from run to run, and for the two configurations).
- Using restarts, 100% of execution time is for `ephwann_shuffle`, of which around 17% is barrier synchronization (mostly in `selfen_elec_q`, but also at finalization), 1% for the `MPI_Allreduce` in `ephwan2blochp`, and of the remaining *Computation* time the bulk is in `rgd_blk_epw` with a lesser amount in `selfen_elec_q`.
- The dramatic load imbalance evident in the initial measurements with 48 MPI processes and a course random grid (where four processes had no work in `rgd_blk_epw`) is no longer present, however, now that there are 216 MPI processes load imbalance remains a significant issue that can be expected to grow with increasing numbers of processes. *Load*

imbalance efficiency for `rgd_blk_epw` is 89%. Time distributions per process are shown in Figure 2, where imbalance in `rgd_blk_epw` and `selfen_elec_q` computation results in significant time in `MPI_Barrier`.

- The computation imbalance of `selfen_elec_q` (called 8000 times by each MPI rank) is correlated to the compute node (each with 24 consecutive MPI ranks). On some compute nodes, all ranks take 2400 seconds, while on other compute nodes, all ranks take approx. 600 seconds.
- Most of the time in `selfen_elec_q` is likely writing of the `linewidth.elself` file (as well as stdout) during the final iteration (based on the previous 48-rank execution trace). Imbalance is accumulated by a subsequent `MPI_Barrier` (green in Figure 2). Although `selfen_elec_q` is executed every iteration, its contribution is expected to be minor.
- The computation imbalance of `rgd_blk_epw` execution time by each rank correlates to the number of calls/visits (Figure 3).
- Six ranks 0,1,2,3,4,7 have 3% more visits than any other ranks, explained by the blockwise distribution of 8000 k-points over 216 ranks resulting in 38 k-points for the first 8 ranks and 37 k-points for the remainder. Despite this small overload, these first 8 ranks aren't the slowest and therefore don't appear to impede the others.
- Approximately 600 (7%) of the 8000 k-points apparently don't result in any significant computational work in `rgd_blk_epw`, and appear to be clustered such that some processes have much less (only one-third as much) work as their peers.
- Barrier synchronisation time in `selfen_elec_q` is anti-correlated to the (imbalanced) *Computation* time in `rgd_blk_epw`. The rank with the longest computation time (number 129) still has 2400 seconds of time in the `selfen_elec_q` `MPI_Barrier` synchronization preceding the `MPI_Allreduce`, which indicates that the load imbalance is not fixed but varying throughout the 8000 iterations.

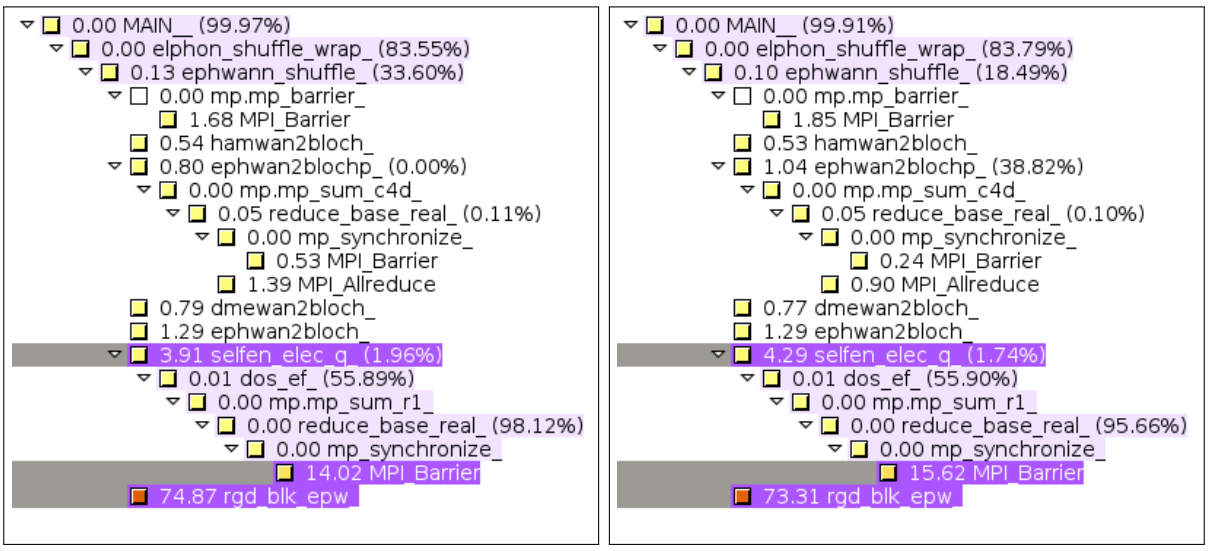

Figure 1: EPW GaN 'default' epw-CB-4q and memory-saving epw-CB-4q-etfmem calltrees showing percentage of total execution time (with 0.5% threshold for hiding).

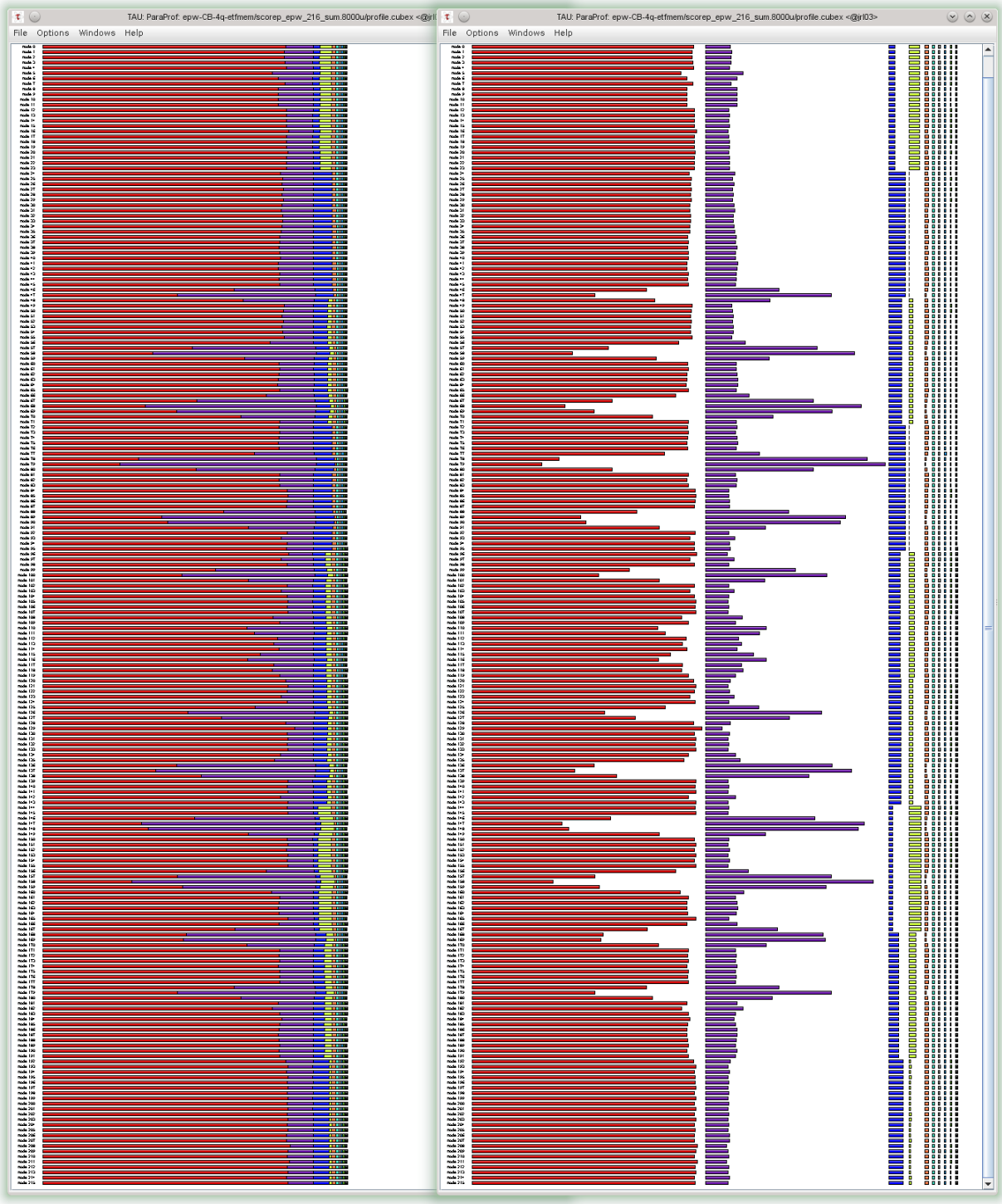

Figure 2: EPW GaN epw-CB-4q-etfmem execution (216 MPI processes on Archer): Stacked chart of exclusive execution time per process on left, unstacked on right. Major components are 73% `rgd_blk_epw` computation (red), 16% `selfen_elec_q` `MPI_Barrier` synchronization (purple), 4% `selfen_elec_q` computation (blue), and 2% concluding `MPI_Barrier` synchronization (green).

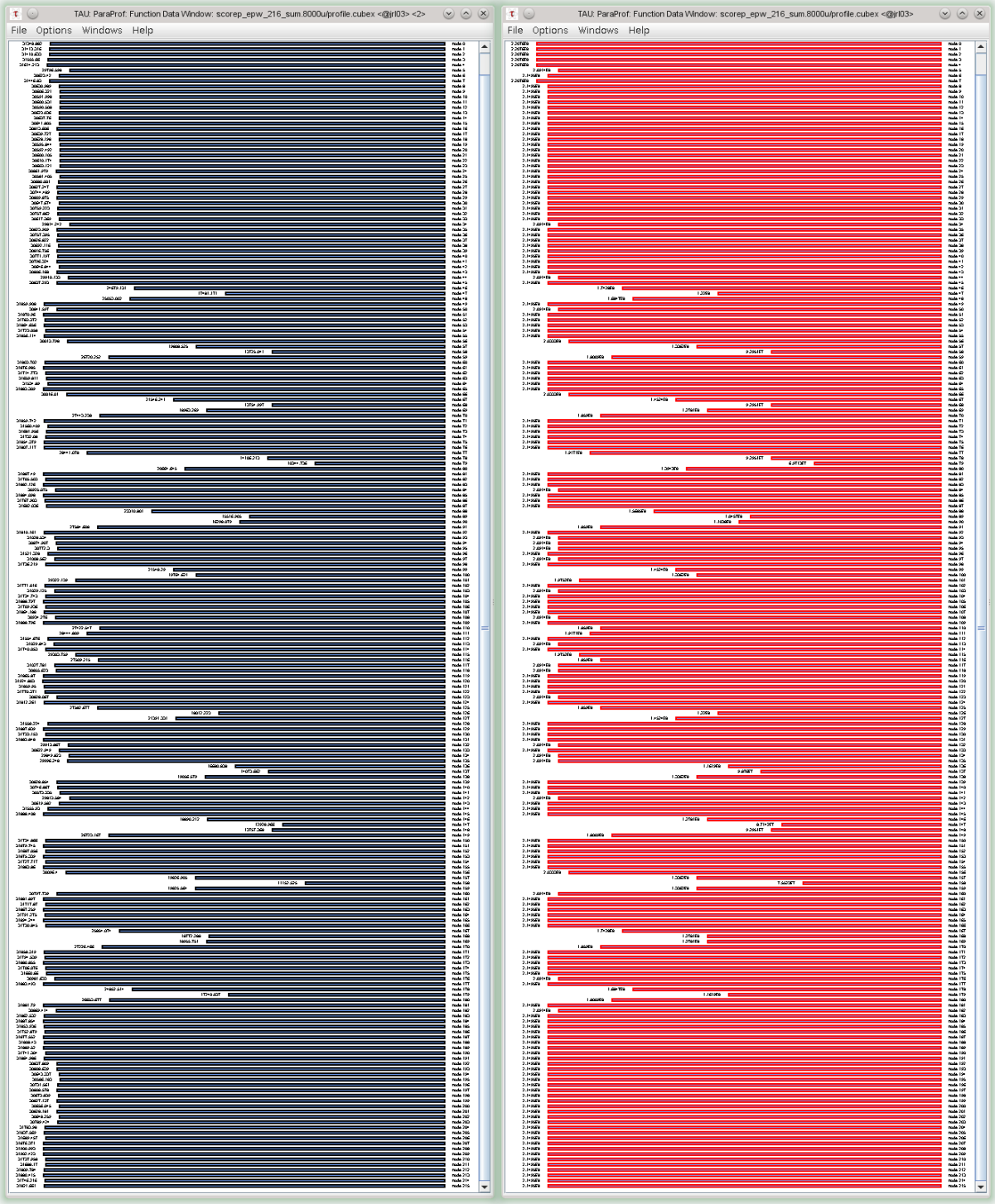

Figure 3: EPW GaN epw-CB-4q-etfmem execution (216 MPI processes on Archer): Histogram of `rgd_blk_epw` exclusive execution time per process on left (black) and corresponding visits/instances on right (red), showing correlation. (Ranks increase from top to bottom.)

3 Analysis of improved load balance (v1)

The previous `rgb_blk_epw` developed for the case of polar materials was replaced by a revised version, `rgb_blk_epw_fine`. Redundant computation in a nested loop for sum over band was eliminated from the specialised version of the routine where it was unnecessary. Also physical reasons allowed a sum over G-vectors to be restricted to a significantly reduced range with no loss of accuracy. Repeating the previous configuration using 216 MPI ranks had execution time reduced 60% from 38923 to 15846 seconds.

Figure 4 shows that the load balance is also significantly improved, with only the first 8 ranks having one extra of the 8000 k-points. While these ranks require a little longer, execution time imbalance is now only 4% compared to 9% previously, and also more than three times faster.

Figure 5 reveals that the 57% of execution time for `rgb_blk_epw_fine` is complemented by a considerably smaller 2.3% for the following `MPI_Barrier` synchronization in `selfen_elec_q`, However, the concluding `MPI_Barrier` synchronization after the final instance of `selfen_elec_q` is now 7% of total time, complementing imbalance in `selfen_elec_q` itself.

Table 1: Parallel efficiency comparison of initial 216-rank configuration in original (216.v0) and revised (216.v1) versions, together with revised version with 480 MPI ranks (480.v1).

Routine	216.v0	216.v1	480.v1
`ephwann_shuffle`	82.36	87.89	71.04
– `rgb_blk_epw`	91.25	96.29	95.34
– `selfen_elec_q`	45.70	76.24	56.94

The comparison of parallel efficiencies of the original and revised versions with 216 MPI ranks in Table 1 confirms that for `rgb_blk_epw_fine` efficiency improved from 91% to 96%, with `ephwann_shuffle` overall improving from 82% to 88% efficiency.

A larger measurement using 27000 k-points with 480 MPI ranks (on 20 compute nodes) showed that although `rgb_blk_epw_fine` load imbalance had grown to 5%, this was still fairly good (with some ranks still having 2% more k-points). The proportion of time for `rgb_blk_epw_fine` had diminished to 32%, however, while `selfen_elec_q` and concluding synchronization had grown to almost 60% of total time. Although the parallel efficiency of `rgb_blk_epw_fine` remained 95%, that of `selfen_elec_q` dropped from 76% to 57% and overall `ephwann_shuffle` down to 71% (Table 1).

It was also observed that the proportional of system CPU time was also large (and variable), hinting that file I/O could be responsible. Although not distinguished in the measurements, the final instance of `selfen_elec_q` includes writing of the final simulation output to file (50MB 'linewidth.elself') and stdout (100MB). The amount of formatted data written is not particularly large, but suggested that the parallel writing was inefficient.

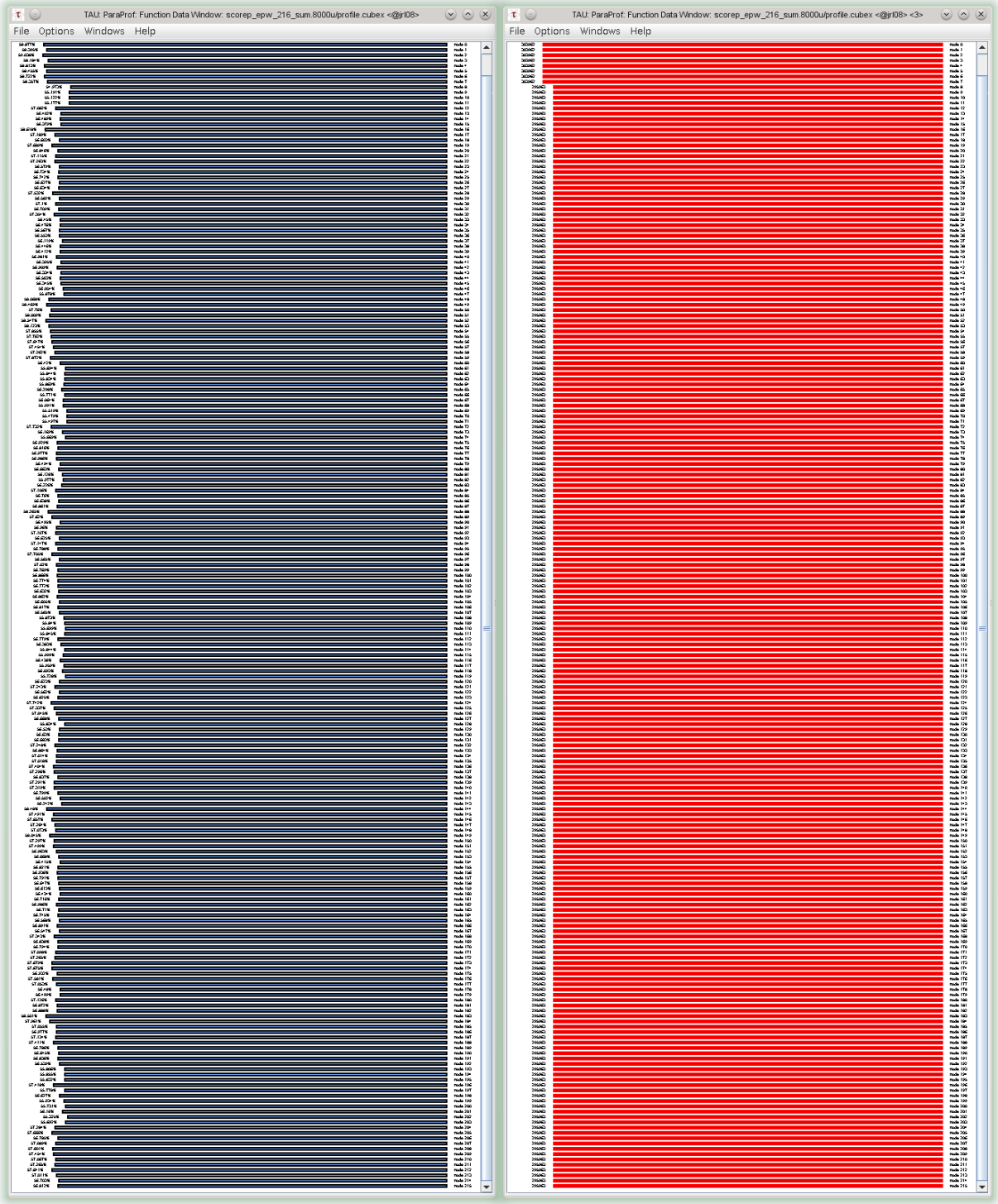

Figure 4: Revision 1 of EPW improving load balance of `rgd_blk_epw_fine` for GaN epw-CB-4q-eftmem execution (216 MPI processes on Archer): Histogram of `rgd_blk_epw_fine` exclusive execution time per process on left (black) and corresponding visits/instances on right (red), showing correlation. (Ranks increase from top to bottom.)

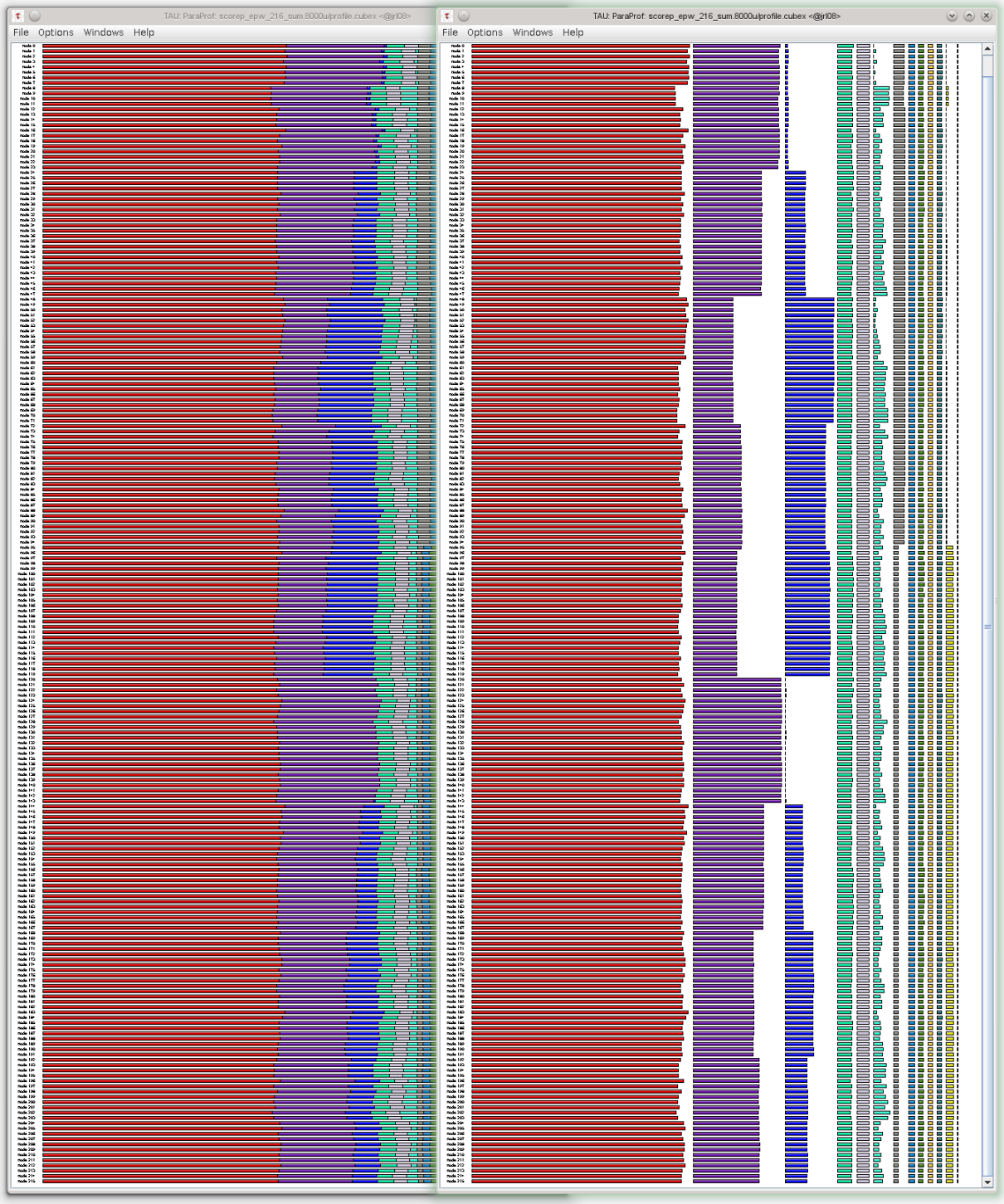

Figure 5: Revision 1 of EPW GaN epw-CB-4q-etfmem execution (216 MPI processes on Archer): Stacked chart of exclusive execution time per process on left, unstacked on right. Major components are now 57% `rgd_blk_epw_fine` computation (red), 17% `selfen_elec_q` computation (purple), and 7% concluding `MPI_Barrier` synchronization (blue). `MPI_Barrier` synchronization in `selfen_elec_q` is now 2.3% (cyan).

4 Summary

This POP Performance Plan of EPW focussed on the load imbalance within `ephwann_shuffle` identified in the prior POP Performance Audit POP_AR_28. Although a finer uniform grid (rather than the previous coarse random grid) was used, with the increased number of MPI ranks — from 48 (on two compute nodes) to 216 (on nine compute nodes) on the Archer Cray XC30 — significant load imbalance of 9% was still observed, manifesting primarily in `rgd_blk_epw`.

A revised version of this routine, `rgb_blk_epw_fine`, specialised to eliminate unnecessary calculation and with optimised vector summations, was 60% faster than the original and had much less imbalance (4%).

EPW could now be used for a larger execution with 480 MPI ranks (on 20 compute nodes) with only modest degradation of load imbalance in `rgb_blk_epw_fine` to 5%. Unfortunately overall performance was somewhat disappointing, as the proportion of time for `rgb_blk_epw_fine` had now diminished to less than one-third of the execution, with the `selfen_elec_q` routine having grown to almost 60%.

EPW `ephwann` execution has 8000 instances of `selfen_elec_q`, however, the final instance is characterised by very different performance, taking much longer and varying substantially according to the compute node on which a process executed. The final execution of `selfen_elec_q` concludes with writing the simulation output to file, and although the amount of data is not large (around 50MB) it becomes a bottleneck inhibiting scaling and larger simulations.

This file writing issue should be investigated in a POP Proof-of-Concept service.