EPW Proof-of-Concept report

Document Information

Reference Number	POP_PoCR_7 (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) uniform fine grid
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.
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 Previous assessment and recommendation

The overall performance of EPW with a GaN/epw-CB-4q test case on the Archer Cray XC30 was reported in POP Performance Audit POP_AR_28. Various load imbalance issues were identified in this original version (v0), and the one considered to be most important in the `ephwann` phase was the subject of a following POP Performance Plan POP_PP_06. Specialisation of the `rgd_blk_epw` routine to eliminate redundant calculation and optimise the summation of G-vectors (version v1) demonstrated significantly improved overall performance and load balance, allowing larger computational simulations to use more processors. Unfortunately, total execution time was disappointing for larger EPW computations, with the `selfen_elec_q` routine having grown to dominate.

This POP Proof-of-Concept study focuses on investigating the nature and origin of the degradation of `selfen_elec_q` performance, incorporating a remedy to improve it, and ultimately to validate scalability to the target configuration size of 1000 processes/cores.

The last instance of `selfen_elec_q` differs from the other instances in that it also writes the final state to a file on disk (50MB of formatted text to 'linewidth.elself') and 100MB to stdout. Although this is not distinguished in function profiles, it was clearly evident in timeline visualisation of execution traces, such as Figure 1.[1] While all MPI processes execute each `selfen_elec_q` instance concurrently, there is a prominent pattern of those processes running on each compute node taking similar amounts of time, and dramatically different times for each compute node (which also varied from run to run). Even in function profiles covering all 8000 instances of `selfen_elec_q`, such as Figure 2,[2] this distinctive pattern by compute node is evident. Highly variable total run times and high proportion of system time also pointed to the likely significance of file I/O.

[1]Vampir display quick reference:
https://pop-coe.eu/sites/default/files/pop_files/vampir_display_quickref.pdf
[2]Cube display quick reference:
https://pop-coe.eu/sites/default/files/pop_files/cube_display_quickref.pdf

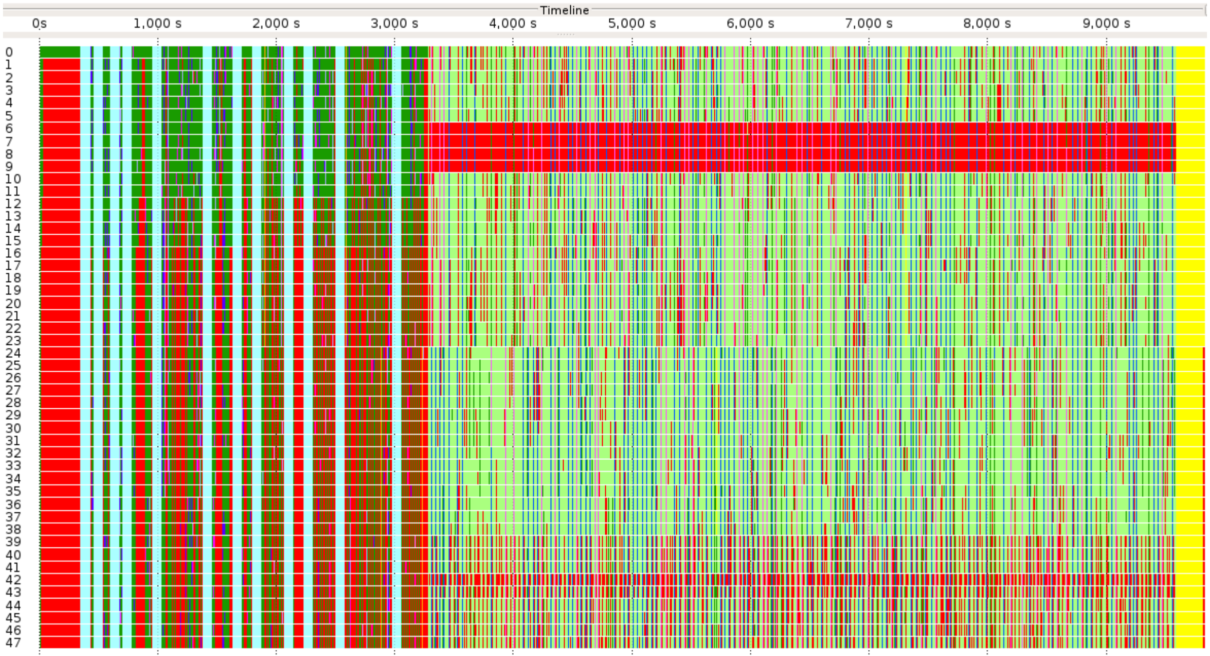

Figure 1: Execution timeline of EPW GaN testcase execution (v0) on two Archer XC30 compute nodes each with 24 MPI ranks (48 MPI processes). `selfen_elec_q` (shown in yellow) executed thousands of times throughout `ephwann_shuffle` phase (pale green), with final instance requiring much longer and unbalanced for processes on each compute node.

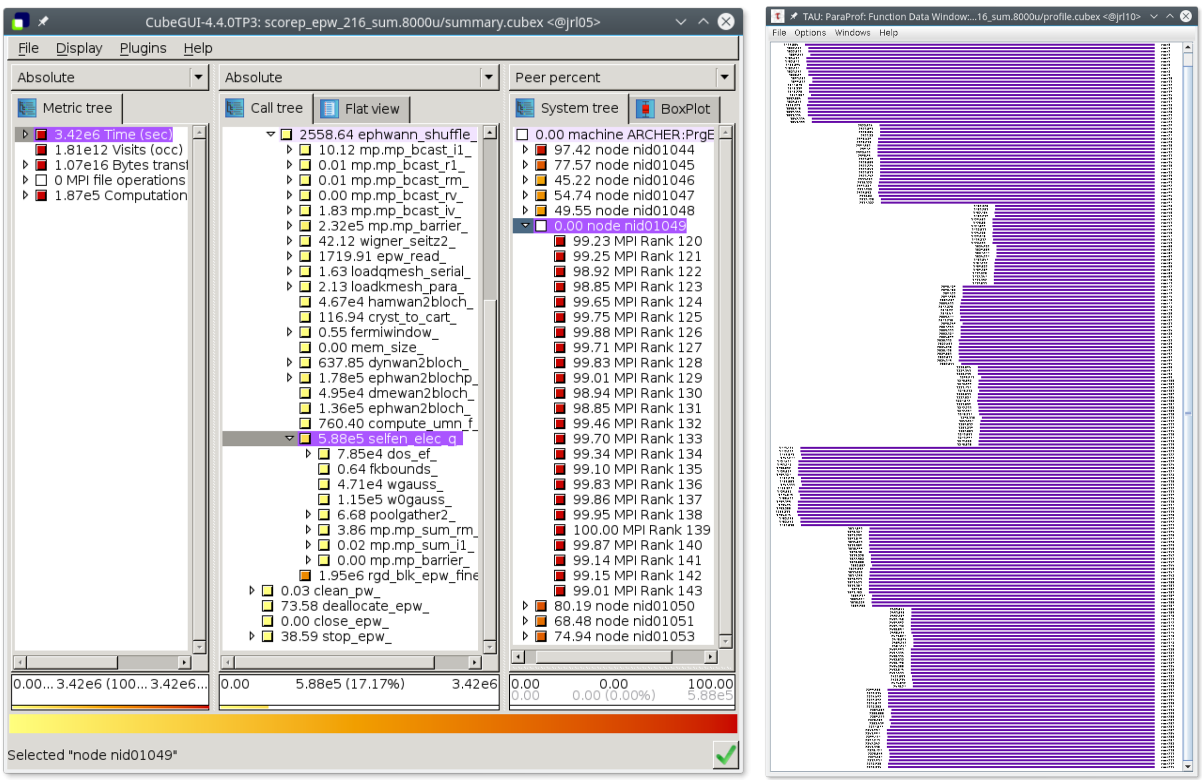

Figure 2: Profile of EPW GaN testcase execution (v1) on nine Archer XC30 compute nodes each with 24 MPI ranks (216 MPI processes). Computation time exclusively in `selfen_elec_q` shows significant variation by process rank with correlation to compute node.

3 Implementation

File writing in `selfen_elec_q` was done concurrently by all MPI ranks, resulting in redundant writing and massive contention for the file on disk: negative performance aspects which grow at least linearly (and potentially worse) with the number of MPI ranks. MPI provides specific routines for (potentially optimised) parallel writing to shared files, and various additional libraries also address parallel file I/O, e.g., parallel HDF5, parallel netCDF and SIONlib. However, since the amount of data being written in this case is relatively small (50MB), the simplest approach for initial investigation would be for only one MPI process (master rank 0) to write the entire data to file. This was straightforward to incorporate with only minor code changes (version v2).

4 Analysis after revised file writing (v2)

With `selfen_elec_q` modified so that the final instance wrote its output data only from rank 0, writing time on 480 MPI ranks reduced from over 7 hours to only 56 seconds: a 450-fold improvement! Subsequent measurements with up to 1920 MPI ranks (80 compute nodes), which were previously unthinkable, also had writing times of less than 60 seconds. Although writing time remains variable from run to run, it is now a negligible component of EPW execution.

Figure 3 shows the scalability of this version of EPW with different numbers of compute nodes of Archer Cray XC30. Execution time is again dominated by `rgb_blk_epw_fine`, which

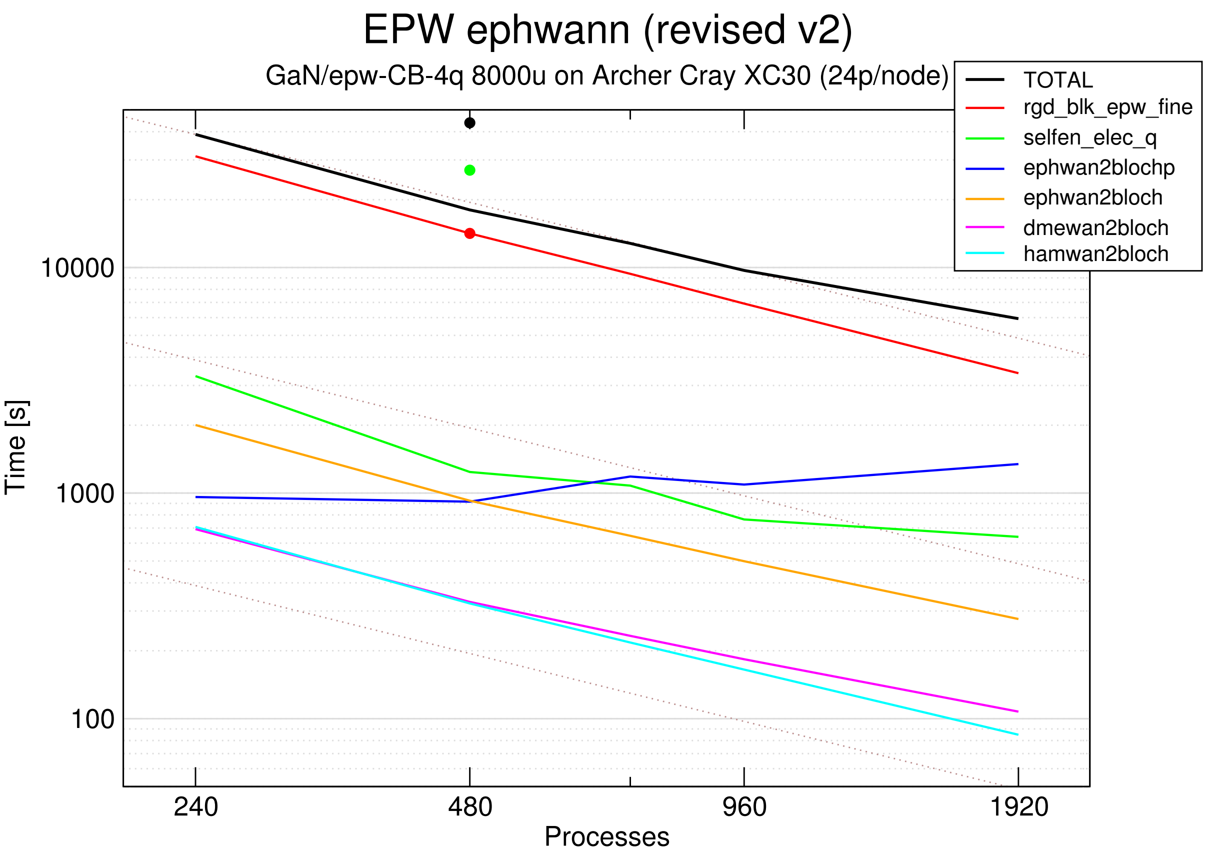

Figure 3: EPW `ephwann` version 2 (GaN epw-CB-4q testcase with 27000 k-points) scalability on Archer Cray XC30. (Previous v1 times with 480 processes for comparison as large dots.)

Table 1: Parallel efficiency comparison of EPW `ephwann` version 2 with improved file writing for different numbers of MPI processes. (Values as percentages)

Routine	240	480	960	1920
`ephwann_shuffle`	92.83	92.16	85.58	71.72
– `rgb_blk_epw_fine`	93.88	94.96	93.02	86.30
– `selfen_elec_q`	88.18	90.53	89.89	89.25
– `ephwan2blochp`	41.12	31.19	23.47	17.95

scales well up to 1920 ranks (80 compute nodes) and is likely to scale further. `selfen_elec_q` (including the revised file writing) and three of the other significant EPW routines also scale well, with the exception being `ephwan2blochp` which requires progressively more time growing to over 22% of the total execution time with 1920 ranks.

For the 8-fold scaling from 240 to 1920 MPI ranks, Table 1 shows moderate reductions of parallel efficiency[3] for `rgb_blk_epw_fine` from 94% to 86% and stable efficiency around 89% for `selfen_elec_q` (excluding the final iteration doing file writing), with overall `ephwann_shuffle` parallel efficiency dropping from 93% to a still quite reasonable 72%. The latter is largely explained by the parallel efficiency of `ephwan2blochp` more than halving from 41% to 18%: both load balance efficiency of 45% and communication efficiency of 40% are major factors for the latter.

[3]POP standard metrics for parallel performance analysis: https://pop-coe.eu/node/69

4.1 1920-process configuration analysis

Focussing on the configuration using 1920 MPI ranks provides insight into the remaining scaling issues. The profile in Figure 4 shows that 28% of total execution time is spent in MPI, with 18.6% for `mp_sum_c4d` within `ephwan2blochp` and 8.6% for `mp_sum_r1` within `selfen_elec_q`. Almost all of the latter is collective synchronization time (and negligible for the 8000 `MPI_Allreduce` calls, one per iteration), whereas in `ephwan2blochp` some 5.1% is collective synchronization time versus 13.5% for the 18 `MPI_Allreduce` calls per iteration. The latter require 800 seconds for the reduction, with MPI ranks above 95 having more than 300 seconds of additional waiting to initiate the `MPI_Allreduce`, largely due to the 95 lowest ranks requiring correspondingly longer for computation in `ephwan2blochp`.

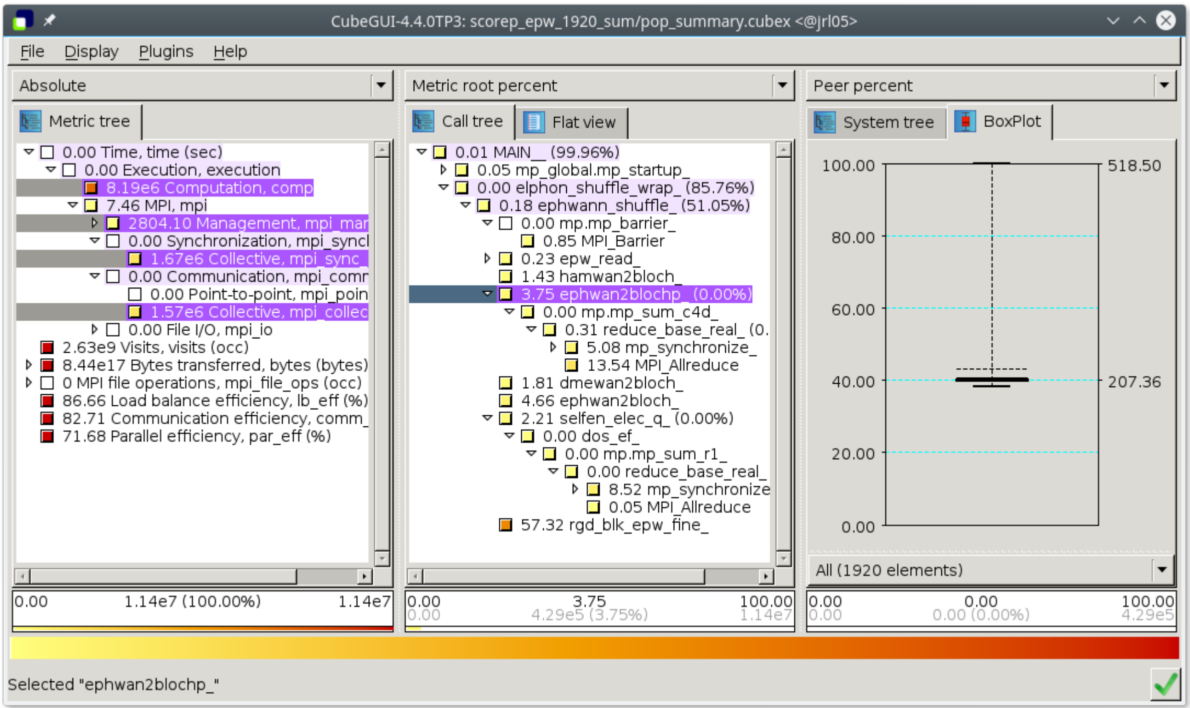

Figure 4: EPW GaN epw-CB-4q execution (1920 MPI processes on Archer): profile of execution time showing computation time in `ephwan2blochp` ranging from around 200 to over 500 seconds, resulting in associated waiting time of 300 seconds on ranks above 95 at following collective synchronization prior to `MPI_Allreduce`.

5 Conclusions

This POP Proof-of-Concept implementation addressed the file writing bottleneck that severely limited EPW performance and scalability when working with large numbers of MPI processes (compute nodes) on the Archer Cray XC30. Previous writing times of many hours have been reduced to under one minute. Whereas use of more than a few compute nodes was previously impractically inefficient, scaling to 1920 MPI processes (80 compute nodes) has now been demonstrated with good parallel efficiency. This also relies on polar material calculation optimisation and associated load balance improvements assessed in the prior POP Performance Plan (POP_PP_06). Both modifications particularly benefit large scale executions of EPW, yet they apply to all computer systems (not just Archer Cray XC30) and also at smaller scale.

While much of EPW `ephwann` seems in good shape to scale to even larger configurations, the originally negligible `ephwan2blochp` routine requires a growing time for collective communication (`MPI_Allreduce`) which will soon dominate. There is also a significant computational imbalance in this routine for a subset of the ranks: further scaling of EPW would need to address this issue.