Implementing a Diffusion Stencil on a Sphere using MPI

Project Report:
High Performance Computing for Weather and Climate (HPC4WC)
ETH Zürich, June 2020

Submitted By: Beat Hubmann, Shruti Nath

Supervised By: Oliver Fuhrer

Background

Atmospheric models consist of a finite volume dynamical core (FV) providing numerical approximations for large-scale atmospheric transport. Such a methodology treats the computational domain as a collection of cells, applying the conservation laws in flux-form across the boundaries of these finite volume cells. Traditionally a latitude-longitude grid has been used to develop these FV algorithms, however, the presence of spherical poles on these grids make achieving highly scalable algorithms difficult. Exploration of other quasi-uniform grids scalable to execution on more computational processors is thus desirable. The cubed sphere grid is one such alternate grid space holding the following advantages:

1) Elimination of pole problem, reducing computation overhead and load imbalance: The clustering of grid points towards the poles in a latitude-longitude grid requires a semi-Lagrangian extension to allow the use of large integration time steps and a polar filter to stabilize the high frequency gravity waves generated at high latitudes. These introduce computation overhead and load imbalances in a parallelized execution. The cubed-sphere grid eliminates these problems at the pole allowing for parallelization and also for a reasonable time step.
2) Spatially resolvable: The cubed-sphere geometry allows for adaptive mesh refinement.

In this exercise, we plan to apply a simple diffusion process onto a cubed-sphere grid and investigate its scalability using Message Passing Interface (MPI) for parallelization. Said cubed-sphere grid consists of six quasi-uniform cartesian coordinate systems representing each face of the cube. A gnomonic projection is used in mapping the cube onto a sphere. Simply put, this entails inscribing the cube within a sphere and expanding it to the surface of the sphere producing a near uniform coordinate system. The six logically rectangular regions of the cubed-sphere are then tiled into 2-D patches for distribution to processing elements (PEs) as schematically shown in Figure 1 below and further elaborated on in the following section.

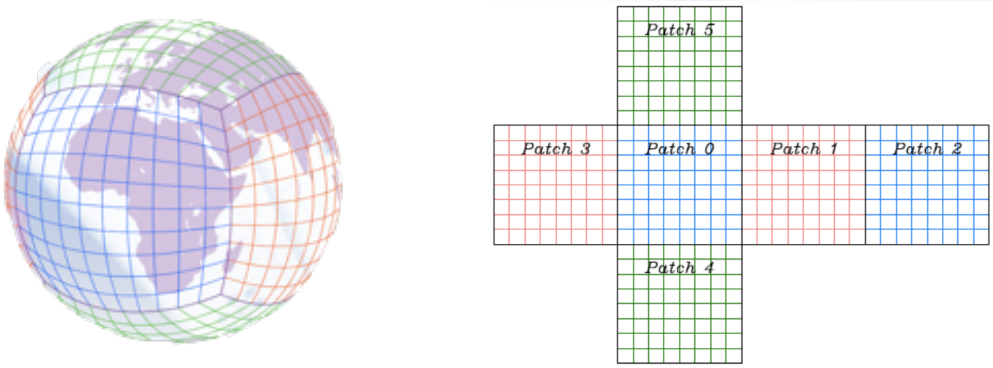

Figure 1: Schematic of cubed sphere grid as applied onto Earth's surface

Implementation

As a first step, a 4th order non-monotonic diffusion stencil was applied over a simple cube surface. Following this, implementation over the cubed sphere grid would have required weighting of grid boxes such that they appear increasingly larger towards a face's centre, however, due to this project's limited scope this refinement step was not carried out. Key to parallelization over the cube surface was a definition of the cube topology such that communication of halo regions across cube faces retained spatial consistency enabling PEs to carry out their work independently.

Topology

In defining the cube topology over which to carry out stencil computations, a 2-D X-Y decomposition of the cube was first applied (shown in the Figure 2 below). Intuitively, this entailed unfolding of the cube, thus enabling tiling of the six faces of the cube into subdomains in the form of 2-D patches. Sharing its parent face's coordinate orientation, each patch then contained a local set of the full cube grid and was assigned an MPI rank.

Tiling and assigning of the cube domain to ranks was implemented in a `CubedSpherePartitioner` class. This class is similar in spirit and function to the `Partitioner` class previously introduced in this HPC4WC course for stencil computation over a 2D grid, however, has additional features dealing with the cubiform topology. These include assigning local ranks besides the global face number as well as storing each patch's orientation relative to its top, bottom, left and right neighbour. Hence, when scattering and gathering the grid space, information of a patch's face number and orientation are retained.

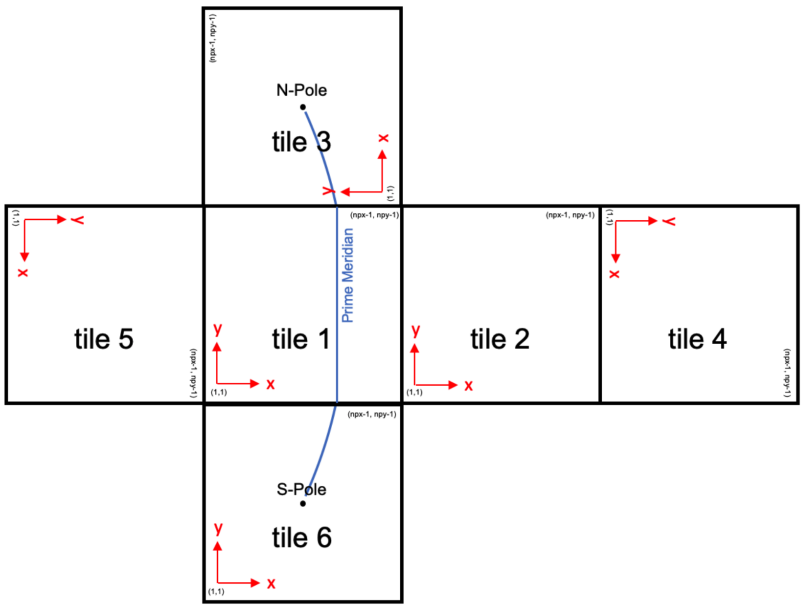

Figure 2: 2D X-Y decomposition of cube

Halo Updates

Each patch carries a halo region on the left, right, bottom and top of the patch. Halo exchanges are required to fill halo regions with updated data for each PE to complete its work independently. In updating halo regions between patches, extra care has to be taken in defining orientation of neighbouring patches. For example, halo exchanges across face edges 2 and 4 where the y-direction scalars on face 2 coincide with x-direction scalars on face 4 would require rotation of the halo region. The curved arrows connecting faces from the outside in Figure 3 below show where flipping of halo x-y axes would be required.

Halo updates were carried out in the `update_halo` function of the *stencil3d-mpi* code. As patches retained their neighbour's orientations relative to themselves, halo regions could be rotated before sending, so as to match the destination patch's axis. As not required by the chosen stencil shape, the corner regions where not included in the halo exchanges to avoid special cases in the eight corners of the cube where three instead of the usual four patches share a point of contact. This was compensated for by filling in information from neighbouring non-corner points when applying the `laplacian` stencil. The numerical errors and potential numerical instability caused by this design decision had to be mitigated by applying a smoothing function to the corner halo points after each halo.

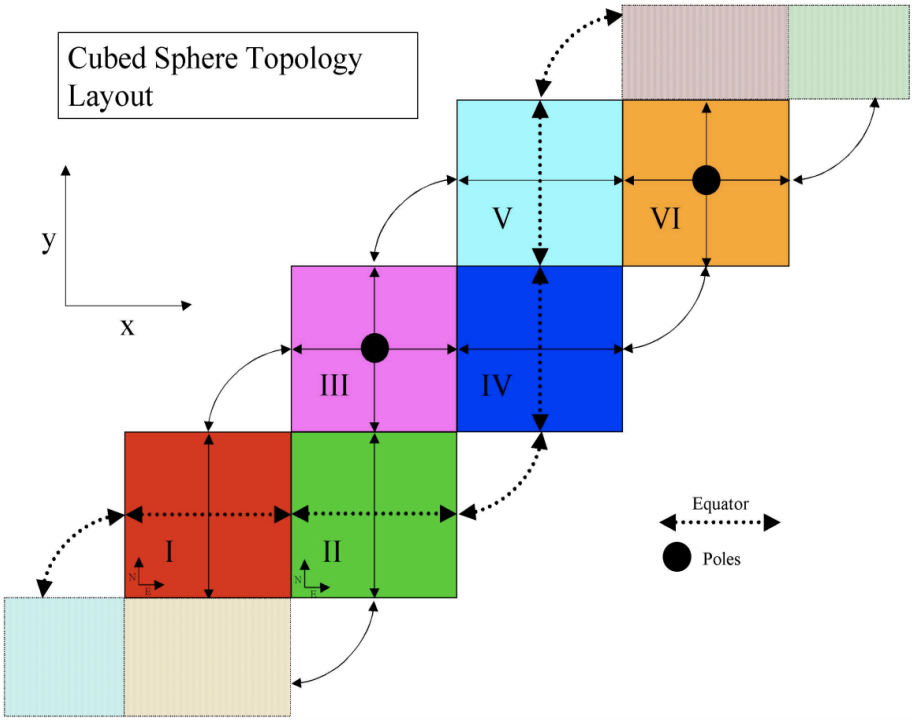

Figure 3: Schematic showing halo exchanges along the cube. Dotted lines move along the equator, and solid lines through the poles. Flipping of halos occur along the curved arrows on the outside of the faces. All faces are oriented along the X-Y direction shown in the top right, although this does not indicate their N-E orientation.

Verification

As the diffusion stencil itself remained as verified during the HPC4WC course. the verification procedure concentrated on checking the halo exchanges. For this, patches were filled with 100*y and x grid values summed up with the rank_value/1000, allowing immediate identification of a grid section's coordinate system orientation and owner rank. For example a point with (y,x) coordinate (30,9) on rank 3 would be given a value 3009.003. The following criteria after a halo update were checked for:

1) Halo originates from neighbour: Done by subtracting the neighbour_rank_value/1000 from the halo values and checking the result had 0 values in the 1st - 4th decimal place
2) Halo does not contain any zero corner field elements indicating a faulty shift: Done by checking that all halo values are > 0
3) Halo is properly oriented along increasing y-axis: Done by checking for gradient>0 along the 1st axis
4) Halo is properly oriented along increasing x-axis: Done by checking for gradient>0 along the 2nd axis

The criteria above were systematically assessed in the top, bottom, left and right halo regions by a `check_halo` function which was called under an assert in a `verify_halo_exchange` function inside the *stencil3d-mpi* code.

Results

Once the halo updates were verified, a visual scan was done to verify the workings of the diffusion stencil over the cube as shown in the Figure 4 below. A `super_plotter` function was used to assemble the patches together as each MPI rank saved its field in a separate `.npy` file labelled with the face number and local rank number.

Figure 4: Infields (on the left, plot includes halo regions) and outfields (on the right, plot without halo regions) of the diffusion stencil applied on the cube. Outfields are plotted after 10,000 iterations.

Scaling Experiments

Having verified that the diffusion on the cube worked as expected, weak and strong scaling experiments were carried out.

Weak scaling analysis was carried out with total MPI ranks of 6, 24, 54, 96, 150 and 216, i.e. 1, 2, 3, 4, 5 and 6 PEs along each of the cube face's axis. Two problem sizes to be kept constant per PE- 60 x 60 x 60 and 120 x 120 x 60-were investigated, with the results shown in Figure 5.

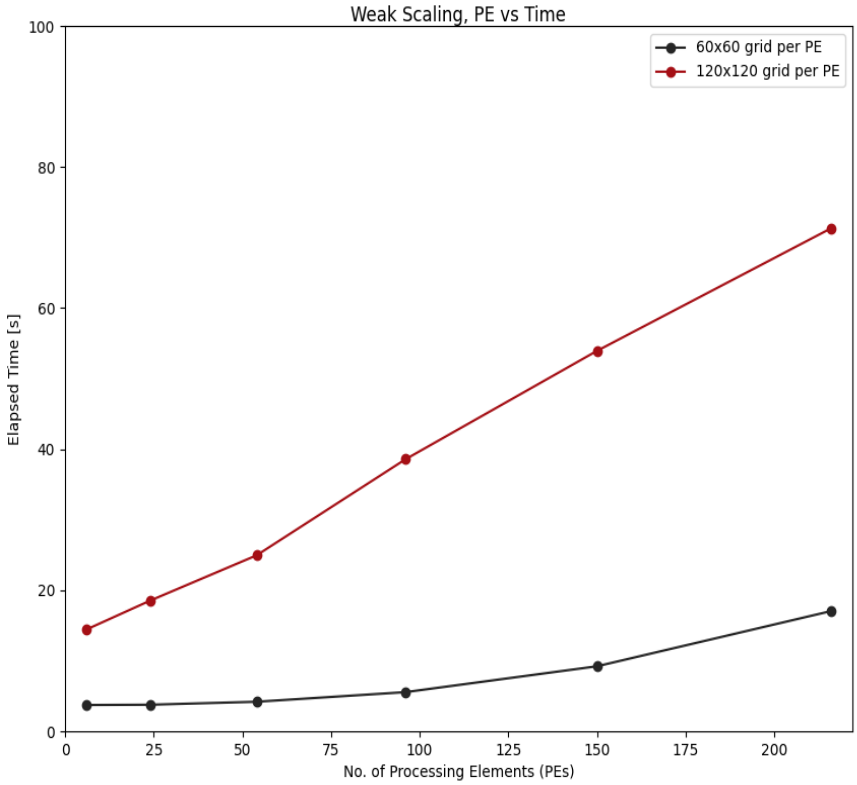

Figure 5: Results of weak scaling experiments. Black and red represent problem sizes of 60x60x60 and 120x120x60 per PE respectively. Elapsed time values are given for 1020 iterations.

From the weak scaling results above, the weak scaling efficiency was calculated as:

$$Scaling\ Efficiency\ for\ N\ PEs = \frac{Elapsed\ Time_{ref\ PEs}}{Elapsed\ Time_{N\ PEs}}$$

The chosen approach not allowing setting the number of overall PE to one as required for a regular scaling analysis, 6 was used as the reference number of PEs (1 per cube face). Scaling efficiencies are shown in Figure 6 below.

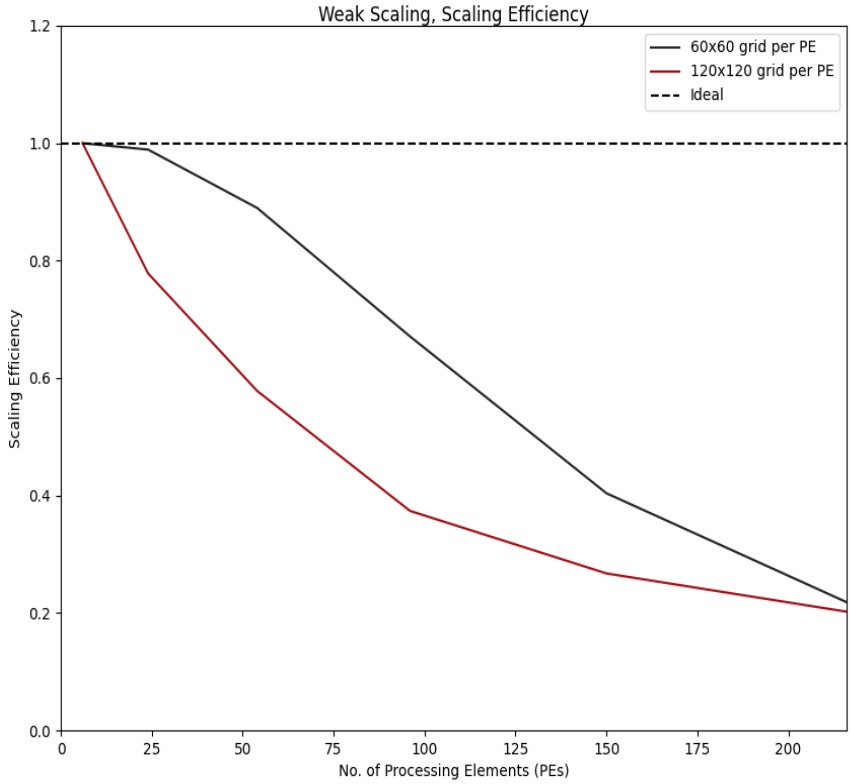

Figure 6: Scaling Efficiency from weak scaling experiments. Black and Red represent problem sizes of 60x60x60 and 120x120x60 per PE respectively. Dashed line is the ideal case.

For the strong scaling experiments fixed problem sizes of 120x120x60 and 240x240x60 were investigated with total MPI ranks of 6, 24, 54, 96, 150 and 216. The results of these are shown in Figure 7 below. The computational speedup with increasing number of PEs was furthermore calculated as:

$$Speedup\ for\ N\ PEs = \frac{Elapsed\ Time_{1\ PE}}{Elapsed\ Time_{N\ PEs}}$$

Results of computational speedup calculations are shown in Figure 8. Strong scaling efficiency was also calculated as $Speedup\ for\ N\ PEs/N$ and plotted in Figure 9.

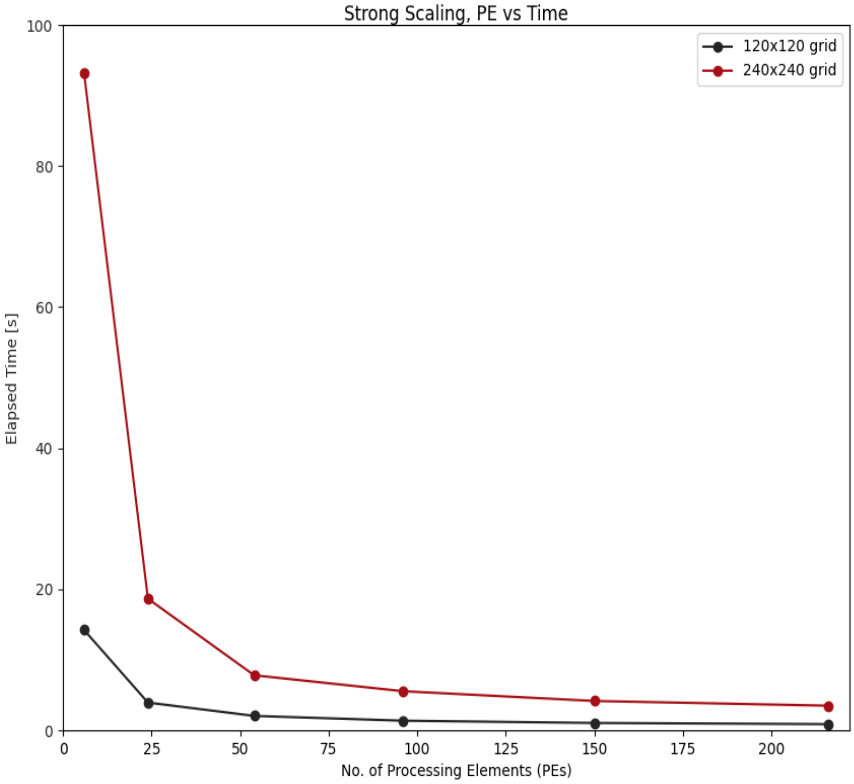

Figure 7: Results of strong scaling experiments. Black and red represent problem sizes of 120x120x60 and 240x240x60 respectively. Elapsed time values are given for 1020 iterations.

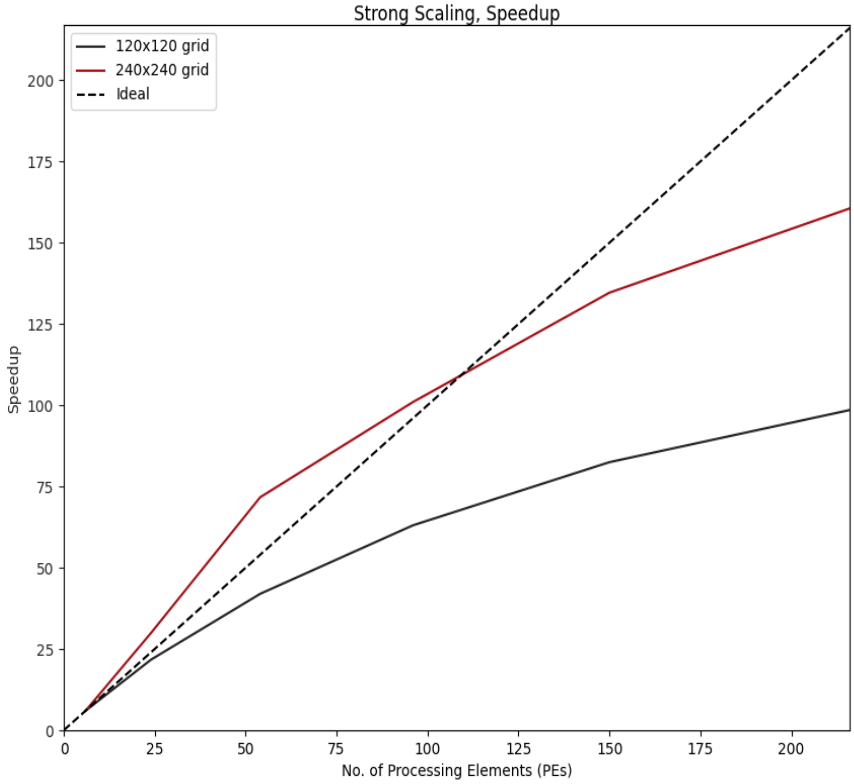

Figure 8: Speedup for strong scaling experiments. Black and Red represent problem sizes of 120x120x60 and 240x240x60 per PE respectively. Dashed line is the ideal case.

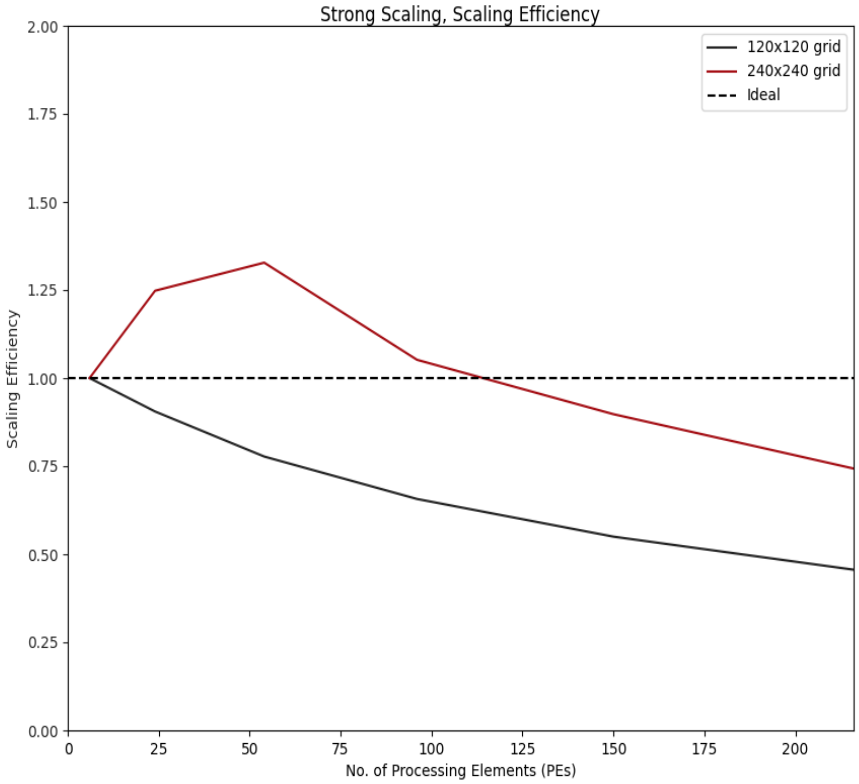

Figure 9: Scaling Efficiency for strong scaling experiments. Black and Red represent problem sizes of 120x120x60 and 240x240x60 per PE respectively. Dashed line is the ideal case.

Discussion

In this exercise, we implemented and verified a diffusion stencil on a cube surface allowing investigation of its scalability using MPI. From the weak scaling results we are able to visualize the communication overhead associated with involving more PEs. In such, the elapsed time shown in Figure 5 increases relatively more after 54 PEs indicating greater communication overhead. This can also be seen by the drop below 0.5 in scaling efficiency for >96 PEs for both 60x60x60 grid per PE and 120x120x60 grid per PE in Figure 6. This is in line with expectations as the chosen implementation did not attempt to optimize communication patterns.

The strong scaling results in Figure 7 indicate a significant speed-up of execution time for a 120x120x60 going from 6 to 96 PEs. After this the performance levels out indicating that we have reached the limit of execution time that can be parallelized with the chosen implementation and are bottlenecked by computations that need to be done serially (as according to Amdahl's law). This reaching of limit is also seen in the levelling out of computational speedup in Figure 8 after an initial improvement from 6 to 96 PEs. The 240x240 resolution scales better, overshooting from ideal at 24 and 54 PEs' as shown in Figures 8 and 9. The overshoot could be caused by several factors including cache utilization and CPU optimization and would require detailed profiling to diagnose.
In conclusion, 96 PE can be established as the sweet spot offering the best compromise between speedup and efficiency for the chosen implementation.

References

Putman, W. M., & Lin, S. J. (2007). Finite-volume transport on various cubed-sphere grids. *Journal of Computational Physics*, *227*(1), 55-78.

Gan, L. (2012) Large-scale Simulation of the Global SWEs on Hybrid CPU-GPU Platforms. NCAR, Boulder, US. Obtainable from: https://data1.gfdl.noaa.gov/multi-core/2012/presentations/Session_6_LinGan.pdf