# Supplementary Material: Loop Tiling in Large-Scale Stencil Codes at Run-time with OPS

I.Z. Reguly, G.R. Mudalige, M.B. Giles

**Abstract**—This document serves as an appendix to the paper "Loop Tiling in Large-Scale Stencil Codes at Run-time with OPS" by Reguly et. al. and provides further performance breakdowns and analysis of the applications being studied.

```
if(dir == g_xdir) {
  if(sweep_number == 1) {
    ops_par_loop(advec_cell_kernel1_xdir, "advec_cell_kernel1_xdir",
                 clover_grid, 2, rangexy,
    ops_arg_dat(work_array1, 1, S2D_00, "double", OPS_WRITE),
    ops_arg_dat(work_array2, 1, S2D_00, "double", OPS_WRITE),
    ops_arg_dat(volume, 1, S2D_00, "double", OPS_READ),
    ops_arg_dat(vol_flux_x, 1, S2D_00_P10, "double", OPS_READ),
    ops_arg_dat(vol_flux_y, 1, S2D_00_0P1, "double", OPS_READ));
  }
  else {
   ops_par_loop(advec_cell_kernel2_xdir, "advec_cell_kernel2_xdir",
                clover_grid, 2, rangexy,
     ops_arg_dat(work_array1, 1, S2D_00, "double", OPS_WRITE),
     ops_arg_dat(work_array2, 1, S2D_00, "double", OPS_WRITE),
     ops_arg_dat(volume, 1, S2D_00, "double", OPS_READ),
     ops_arg_dat(vol_flux_x, 1, S2D_00_P10, "double", OPS_READ));
  }
  ops_par_loop(advec_cell_kernel3_xdir, "advec_cell_kernel3_xdir",
               clover_grid, 2, rangexy_inner_plus2x,
    ops_arg_dat(vol_flux_x, 1, S2D_00, "double", OPS_READ),
    ops_arg_dat(work_array1, 1, S2D_00_M10, "double", OPS_READ),
    ops_arg_dat(xx, 1, S2D_00_P10_STRID2D_X, "int", OPS_READ),
    ops_arg_dat(vertexdx, 1, S2D_00_P10_M10_STX, "double", OPS_READ),
    ops_arg_dat(density1, 1, S2D_00_P10_M10_M20, "double", OPS_READ),
    ops_arg_dat(energy1, 1, S2D_00_P10_M10_M20, "double", OPS_READ),
    ops_arg_dat(mass_flux_x, 1, S2D_00, "double", OPS_WRITE),
    ops_arg_dat(work_array7, 1, S2D_00, "double", OPS_WRITE));
...
```

**Fig. 1:** Example of CloverLeaf's high-level source

```
inline void advec_mom_kernel1_x_nonvector(
    const double *node_flux, const double *node_mass_pre,
    double *mom_flux, const double *celldx, const double *vel1) {
  double sigma, wind, width;
  double vdiffuw, vdiffdw, auw, adw, limiter;
  int upwind, donor, downwind, dif;
  double advec_vel_temp;
  if( (node_flux[OPS_ACC0(0,0)]) < 0.0) {
    upwind = 2;
    donor =1;
    downwind = 0;
    dif = donor;
  }
  else {
    upwind=-1;
    donor=0;
    downwind=1;
    dif=upwind;
  }
  sigma = fabs(node_flux[OPS_ACC0(0,0)])/
              node_mass_pre[OPS_ACC1(donor,0)];
  width = celldx[OPS_ACC3(0,0)];
  vdiffuw = vel1[OPS_ACC4(donor,0)] - vel1[OPS_ACC4(upwind,0)];
  vdiffdw = vel1[OPS_ACC4(downwind,0)] - vel1[OPS_ACC4(donor,0)];
  limiter=0.0;
  if(vdiffuw*vdiffdw > 0.0) {
    auw = fabs(vdiffuw);
    adw = fabs(vdiffdw);
    wind = 1.0;
    if(vdiffdw <= 0.0) wind = -1.0;
    limiter=wind*MIN(width*((2.0-sigma)*adw/width+(1.0+sigma)*
            auw/celldx[OPS_ACC3(dif,0)])/6.0, MIN(auw, adw));
  }
  advec_vel_temp = vel1[OPS_ACC4(donor,0)] + (1.0 - sigma) * limiter;
  mom_flux[OPS_ACC2(0,0)] = advec_vel_temp * node_flux[OPS_ACC0(0,0)];
}
```

**Fig. 2:** Example of CloverLeaf's kernel code

## Appendix A
## Code snippets

CloverLeafis a substantially more complex application than the polyhedral compiler benchmark codes in several respects: it consists of multiple compilation units, there are many code paths even in a single source file that affects the sequence of loops as shown in Figure **??**, and the stencils are sometimes data-dependent, as shown in Figure **??**

## Appendix B
## Tile sizes

In the main paper, we refer to exhaustively searching for the tile size that gives the best performance - there we only report on the best one as well as the tile size determined automatically by OPS.

Figure **??** shows performance achieved by Pluto, our hand-coded benchmark and OPS at different tile sizes and heights. In Figure **??**(a) for Pluto, we show a slide of the optimisation space where the X tile size is fixed at 112, and we vary the Y tile size as well as the tile height (the number of time iterations timed over) - the best performance is achieved at $112 \times 32$ and a height of 32. In Figure **??**(b) for the hand-coded benchmark the tile size in the X direction is always 8192, and we show performance achieved at various Y tile sizes and tile heights, the best performance is achieved at a Y size of 120 and a tile height of 50. Finally, in Figure **??**(c) for OPS, we also show a slice of the optimisations space where the X tile size is fixed at 8192, the best performance is achieved at a Y tile size of 100, and a height of 30.

We carried out a similar exhaustive search over possible X and Y tile sizes for CloverLeaf 2D and 3D, however here we do not change the tile height - it is always as "tall" as all the loops in a single time iteration, which is 153 in 2D and 603 in 3D. Figure **??** shows the performance on CloverLeaf 2D: while the best performance is achieved at a tile size of

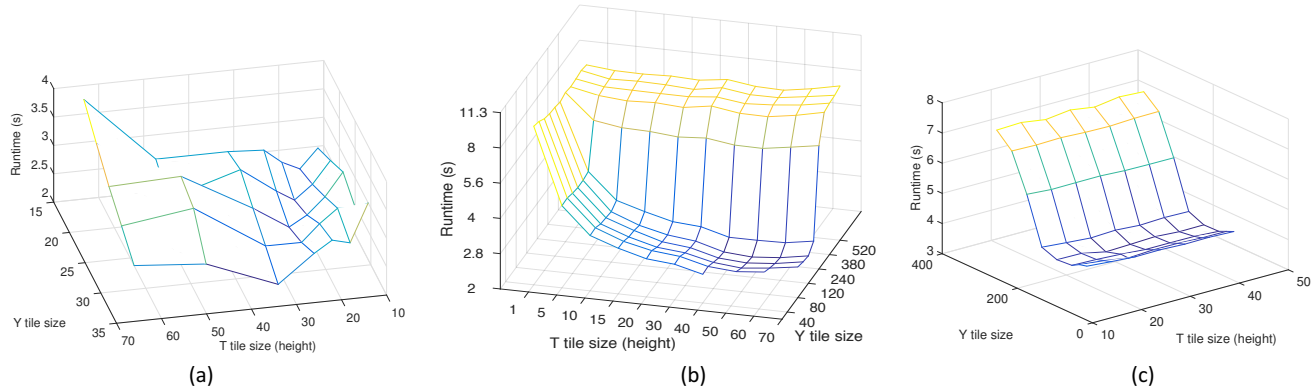


**Fig. 3:** Performance of tiling approaches on a $8192^2$ 2D Jacobi problem over 250 iterations, varying Y tile size and number of iteration tiled. (a) Pluto, (b) handcoded, (c) OPS.

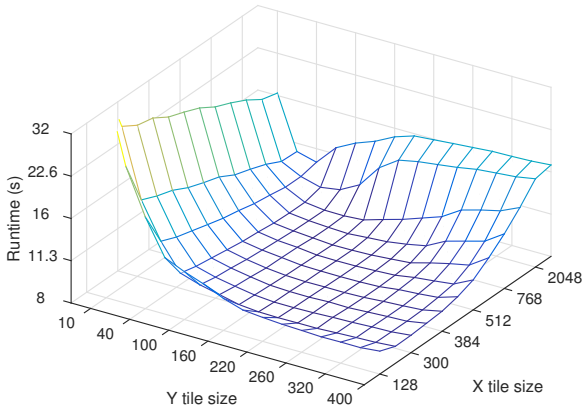


**Fig. 4:** Performance of OPS on CloverLeaf 2D with a $6144^2$ mesh over 10 iterations

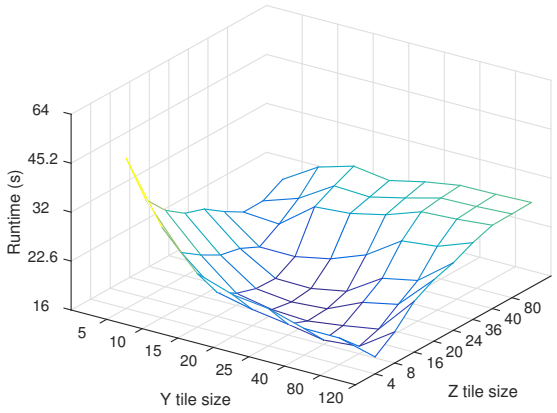


**Fig. 5:** Performance of OPS on CloverLeaf 3D with a $330^3$ mesh over 10 iterations

$640 \times 160$, there are a large number of possible size combinations (32) that are within 2% of the best performance. Figure **??** shows performance at different tile sizes on CloverLeaf 3D, when the X tile size is fixed at 330: here due to the restriction that the product of the Y and the Z tile sizes has to be an integer multiple of the number of OpenMP threads, the number of possible combinations is lower than in the 2D case. The best performance is achieved at a tile size of $330 \times 20 \times 20$, but 6 other possible combinations were within 2% of the best, and 18 within 10%.

## APPENDIX C
## TILE "HEIGHT" - LOOPS TILED OVER

Due to the diversity of loops in the applications being studied, setting the number of loops to tile across to a fixed value does not make sense - instead we choose boundaries between computational stages. For CloverLeaf 2D and 3D the code structure is illustrated in Figure **??** - we mark the natural boundary, due to a reduction, between time iterations with red, and we introduce additional boundaries marked by points 1-5. In the code, these are simply API calls that trigger the tiled execution of loops queued up to that point. We ran a series of tests enabling and disabling

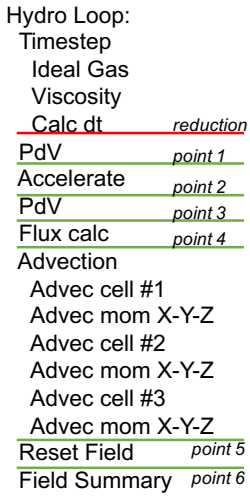


**Fig. 6:** Code structures of CloverLeaf 2D and 3D

these boundaries in all possible permutations ($2^6$), yielding different tile "heights", that is the number of loops to be tiled across. Performance at different permutations listed lexicographically (i.e. 0 is none, 1 is point 1, 2 is point 2, 3 is points 1 and 2, etc.) is shown in Figures **??** and **??**.

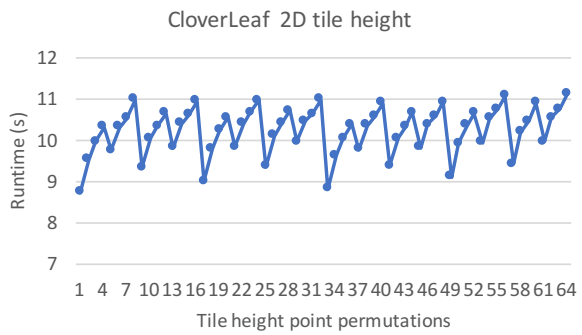


**Fig. 7:** Performance when tiling across different numbers of loops in CloverLeaf 2D

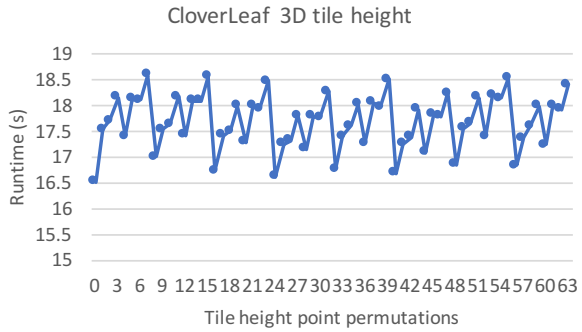


**Fig. 8:** Performance when tiling across different numbers of loops in CloverLeaf 3D

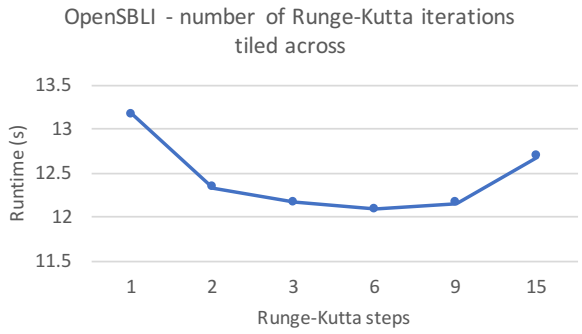


**Fig. 9:** Performance when tiling across different numbers of loops in OpenSBLI

Performance close to the best is also achieved when point 4 and point 5 are both enabled. Performance on OpenSBLI when tiling across increasing number of Runge-Kutta steps (one timestep is 3 RK steps) is shown in Figure **??** - best performance is achieved when tiling over all loops in two time steps.

## Appendix D
## OpenMP Scaling

We also carried out a standard benchmark when using multi-threading, OpenMP in our case: thread scaling. Figure **??** shows performance of the baseline and tiled versions of CloverLeaf 2D. As expected, performance scales up to the number of physical cores, and there is a little improvement

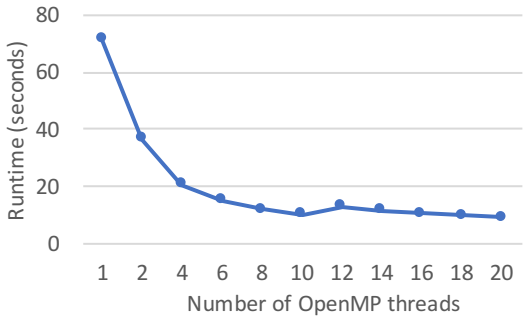


**Fig. 10:** Performance of CloverLeaf 2D with a $6144^3$ mesh with different numbers of OpenMP threads

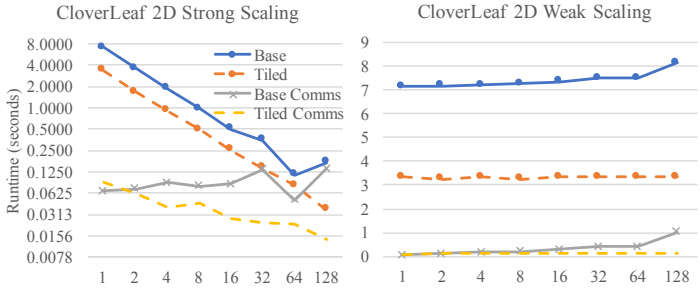


**Fig. 11:** Scaling CloverLeaf 2D to multiple nodes on Marconi

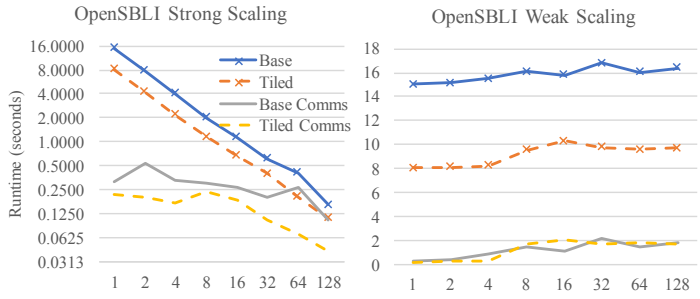


**Fig. 12:** Scaling OpenSBLI on Marconi

by using Hyper-Threading. As it commonly happens with bandwidth-bound applications, the performance improvement slows down as the number of threads is increased, because even a smaller number of threads can saturate the available bandwidth.

## Appendix E
## MPI Scaling on CPUs

Here we present and discuss the scalability of our baseline (MPI+OpenMP) and tiled versions on the applications not shown in the main paper. Figure **??** shows strong and weak scaling of CloverLeaf 2D on a $6144^2$ mesh. In both cases the $2.2\times$ performance improvement is maintained up to high core counts, and particularly on strong scaling we can see the benefit of our improved communications scheme. Figure **??** shows similar results on OpenSBLI, when scaling a $257^3$ mesh; the tiled version maintains a $1.7\times$ speedup over the baseline

## APPENDIX F
## MPI SCALING ON KNL

Figure **??** shows MPI scaling on the Knights Landing platform, with CloverLeaf 2D and the $6144^2$ mesh, which does fit in the 16GB cache, and a $14336^2$ mesh, which does not fit in the 16GB cache. When strong scaling the $14336^2$ mesh, there is initially a $1.45\times$ speedup on a single node, but moving to 2 or more nodes, the problem does fit in the 16GB cache, and there is no improvement in terms of computational time, however, there is a significant benefit from the improved communications scheme. Similar effects can be observed when weak scaling - on the larger mesh, we maintain a $1.5\times$ improvement due to the cache optimisation, but for both meshes, the communication times are improved with the tiled version. During weak scaling, unlike for CloverLeaf 3D, the cost of communications does not increase as much, due to a better ratio of boundary to full volume.

Figure **??** shows MPI scaling on KNL and OpenSBLI, using a $390^3$ mesh (fitting in 16GB) and a $540^3$ mesh (which does not fit in 16GB). Scaling is very similar to that of CloverLeaf 3D, with the tiled version scaling much better than the baseline, largely due to improved communications, as well as cache efficiency in the $540^3$ weak scaling case.

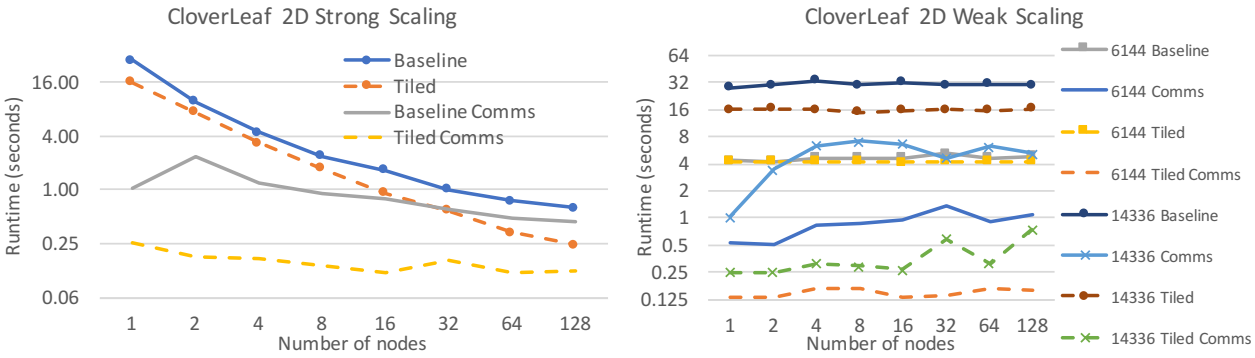


**Fig. 13:** Scaling CloverLeaf 2D on Marconi-A2 (KNL)

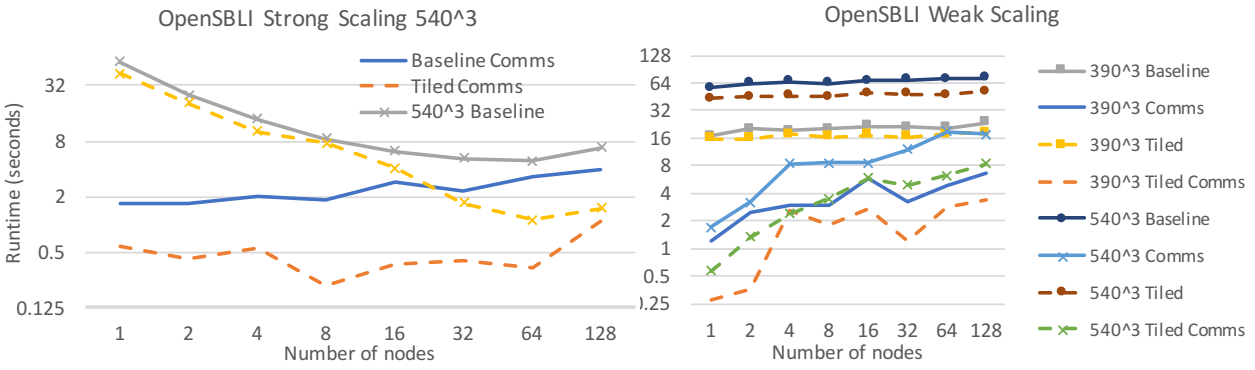


**Fig. 14:** Scaling OpenSBLI on Marconi-A2 (KNL)