PCAM Process:

Partitioning:

- I wanted a `source` processor to (1) initialize the starting matrix and (2) print out the calculated matrix at iteration steps 0, 20, 40, and 80. I then wanted all processors to implement the rules of the Game of Life in each of their local matrices.

Communication:

- (Game of Life: Parallel - Column Decomposition) Since I needed the right column of the left processor and the left column of the right processor to wrap each local matrix, I needed to have 2 neighboring processors communicating.
- (Game of Life: Parallel - 2D Decomposition) Since I needed the right column of the left processor, the left column of the right processor, the bottom row of the top processor, the top row of the bottom processor, and all corresponding corners to wrap each local matrix, I needed to have 8 neighboring processors communicating.

Agglomeration:

- (Game of Life: Parallel - 2D Decomposition) Instead of having 8 neighboring processors communicating, I instead had 4 communication channels for the right column of the left processor, the left column of the right processor, the bottom row of the top processor, and the top row of the bottom processor and then inferred the corners needed using that information[1].

Mapping:

- (Game of Life: Parallel - Column Decomposition) Each processor will be given `n/numprocs` number of columns to create sub matrices; if `mod(n, numrpocs) .ne. 0` then the number of columns will be increased by 1 and then all extra columns in the end processor will be dropped. Finally, all processors will be implementing the rules of the Game of Life, but the `source` processor will be the "main/reporting" processor.
- (Game of Life: Parallel - 2D Decomposition) Each processor will be given an evenly divided sub matrix[2], and again, all processors will be implementing the rules of the

[1] See subroutine `periodicMat` in `GOL2D.f90` to see how I implemented this.

[2] I imposed the idea that the starting matrix *had* to break off into square sub matrices given the number of processors; if it did not, or if the sub matrix dimensions did not add up to the starting matrix's dimensions, then the code would stop there.

Game of Life, but the `source` processor will be the "main/reporting" processor.

Caveats

(Game of Life: Parallel - 2D Decomposition) Unfortunately, this code does not work *completely*. While, yes, it does correctly scatter and gather even sub matrices to and from each processor, it does not complete the Game of Life correctly as the last iteration that is correct is iteration 20. I *think* this has to do with how I define the processor space, or my method of deriving the corners was wrong, but I'm not sure. In any case, sorry Nic and Winson!

Game of Life: Sequential

The sequential version of the Game of Life can be found in `GOLseq.f90`. To create its respective executable type `gfortran GOLseq.f90 -o GOLseq.ex`. To run the executable type `./GOLseq.ex`. Finally, a sample output can be found in `output.txt`, but typing the aforementioned command should produce the same result.

Game of Life: Parallel - Column Decomposition

The parallel version of the Game of Life using column decomposition can be found in `GOLcol.f90`. To create its respective executable type `mpif90 GOLcol.f90 -o GOLcol.ex`. To run the executable type `sbatch GOLcol.cmd`. Finally, the output can be found in `GOLcol.out`.

Game of Life: Parallel - 2D Decomposition

The parallel version of the Game of Life using 2D decomposition can be found in `GOL2D.f90`. To create its respective executable type `mpif90 GOL2D.f90 -o GOL2D.ex`. To run the executable type `sbatch GOL2D.cmd`. Finally, the output can be found in `GOL2D.out`.

Overall, to change the rows (`m`), columns (`n`), and iterations (`iter`), change the first few lines of code on the Fortran file(s) of interest. To change the number of processors for the parallel versions, see all `.cmd` files. By default, all Fortran files have `m = 20`, `n = 20`, `iter = 80`, and `numprocs = 4`.

Note: I created this code on UCSC's `Hummingbird` supercomputer, not UCSC's `grape`

supercomputer. Type `mpirun -np 4 GOLcol.ex` and `mpirun -np 4 GOL2D.ex` to run executables on `grape`.

Performance Model

Let $N_x \times N_y$ denote a 2D grid of data points.
Case 1: Game of Life: Parallel - Column Decomposition (2 neighboring processors)
Then, we have that each P task is responsible for $N_y \times 1$ points and must exchange N_y points with 2 neighboring P tasks. Hence,

$$\begin{aligned} T_{comp} &= t_c N_x N_y, \qquad \text{(No replicated computation)} \\ T_{comm} &= 2P(t_s + t_w N_y), \\ \implies T_{col} &= \frac{t_c N_x N_y}{P} + 2t_s + 2t_w N_y, \\ \implies E &= \frac{t_c N_x N_y}{t_c N_x N_y + 2Pt_s + 2Pt_w N_y}. \end{aligned}$$

Subsequently, for constant efficiency, we require

$$\begin{aligned} t_c N_x N_y &= E(t_c N_x N_y + 2Pt_s + 2Pt_w N_y), \\ t_c N_y &= E(t_c N_y + 2t_s + 2t_w N_y). \qquad (N_x = P) \end{aligned}$$

Thus, the isoefficiency of this algorithm is $\sim O(P^2)$.
Case 2: Game of Life: Parallel - 2D Decomposition (8 neighboring processors)
Then, we have that each P task is responsible for $(N_x/\sqrt{P}) \times (N_y/\sqrt{P})$ points and must exchange $N_x/\sqrt{P}$ points with 2 neighboring P tasks, $N_y/\sqrt{P}$ points with 2 neighboring P tasks, and 1 point with 4 neighboring P tasks. Hence,

$$\begin{aligned} T_{comp} &= t_c N_x N_y, \qquad \text{(No replicated computation)} \\ T_{comm} &= 2P(t_s + t_w N_x/\sqrt{P}) + 2P(t_s + t_w N_y/\sqrt{P}) + 4P(t_s + t_w), \\ \implies T_{2D} &= \frac{t_c N_x N_y}{P} + 8t_s + 2t_w N_x/\sqrt{P} + 2t_w N_y/\sqrt{P} + 4t_w, \\ \implies E &= \frac{t_c N_x N_y}{t_c N_x N_y + 8Pt_s + 2\sqrt{P}t_w N_x + 2\sqrt{P}t_w N_y + 4Pt_w}. \end{aligned}$$

Subsequently, for constant efficiency, we require

$$\begin{aligned} t_c N_x N_y &= E(t_c N_x N_y + 8Pt_s + 2\sqrt{P}t_w N_x + 2\sqrt{P}t_w N_y + 4Pt_w), \\ t_c &= E(t_c + 8t_s + 2t_w + 2t_w + 4t_w). \qquad (N_x = N_y = \sqrt{P}) \end{aligned}$$

Thus, the isoefficiency of this algorithm is $\sim O(P)$. Therefore, in comparing this to the results found in the column decomposition case, we may conclude that a 2D decomposition is more scalable than a column decomposition.

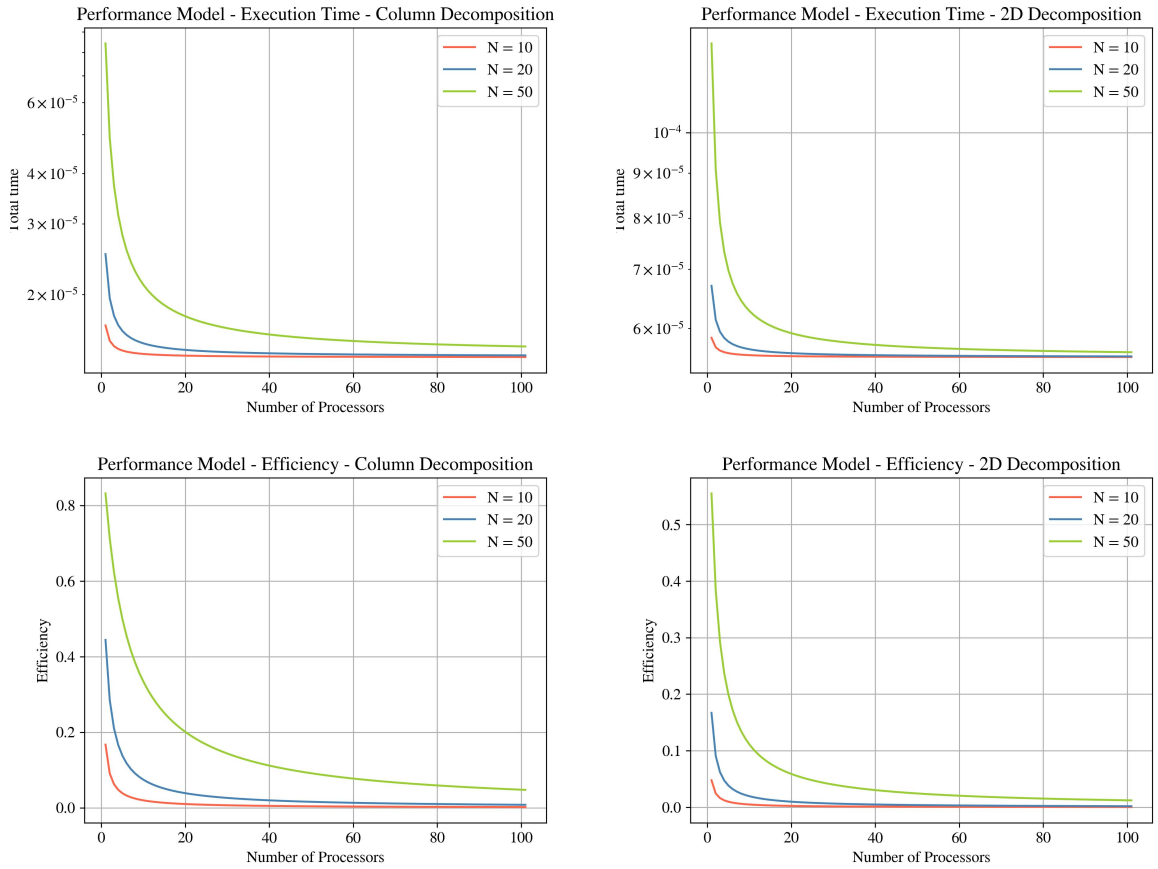

Figure 1: Plots of T and E as a function of P and N ($t_c = 2.8E{-}008$, , $t_s = 6.96E{-}006$, $t_w = 2.58E - 009$)