Game of Life Report

Bryan Garcia
AM 250

June 8, 2020

1 PCAM Analysis

Recall our four main weapons of **PCAM**: *partitioning*, *communication*, *agglomeration*, and *mapping*.

1.1 Partitioning

The first decision was to design the parallel algorithm with *domain* decomposition in mind. By partitioning the $N \times N$ grid by column, we can recognize concurrency in the ability to apply the GOL rules on each column independently.

1.2 Communication

Because we have decided to partition the domain by column, we must establish a communication pattern amongst adjacent columns. To do so, the "ring" communication structure was adopted. That is, each column will communicate with a column to their "left" and "right." The first column's "left" will be the last column, and the last column's "right" will be the first column.

1.3 Agglomeration

Agglomeration of a column-based domain decomposition with "ring" communication can begin with grouping multiple columns. More columns within a group reduces communication costs and allows for tasks to be executed without interruption (e.g. sending and receiving data).

1.4 Mapping

Load balancing processors can be achieved by distributing an equal number of columns to each processor. Ideally, we should have $\mod(N, p) = 0$ and $\frac{N}{p}$ many columns per processor. Should the remainder not be zero, we assign the excess to one of the processors. This mapping configuration only requires communication between columns when updating the state of the GOL grid.

2 Performance Model

The following performance model was created to reflect a column-wise domain decomposition.

Our goal is to create an estimate for our total GOL execution time T_{GOL}. Define this as:

$$T_{GOL} = \frac{T_{comp} + T_{comm}}{P}$$

Suppose our grid is a square with height and width of size N. With column-wise domain decomposition and by assuming no replicated duplication, let us say

$$T_{comp} = t_c N$$

with t_c as the computation time on a *single* element of our grid. The quantity above should represent the computation time on a single column. Now let us consider communication time. For this, say:

$$T_{comm} = 2P(t_s + t_w(N + 2))$$

with t_w as our write time and t_s as our startup time. Note that we are operating on $(N+2)$ columns. This is in order to take into account ghost columns that must be communicated between processors. Since we are employing a "ring" communication pattern, we must transmit data to our left and right. This justifies the factor of $2P$ in our T_{comm} estimate. All together, we obtain:

$$T_{GOL} = 2(t_s + t_w(N + 2)) + \frac{t_c N}{P}$$

My latency constants were obtained from the data I have visualized below:

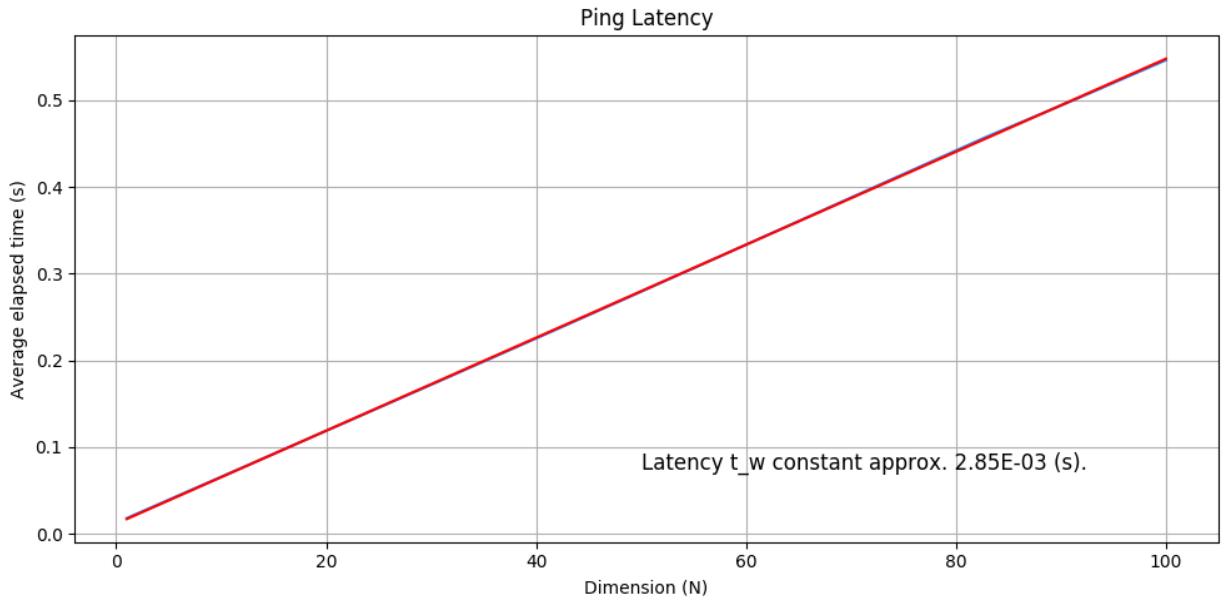

Figure 1: Older latency plot from `ping.f90` code. Not pictured: latency constant t_s = 1.10E-02.

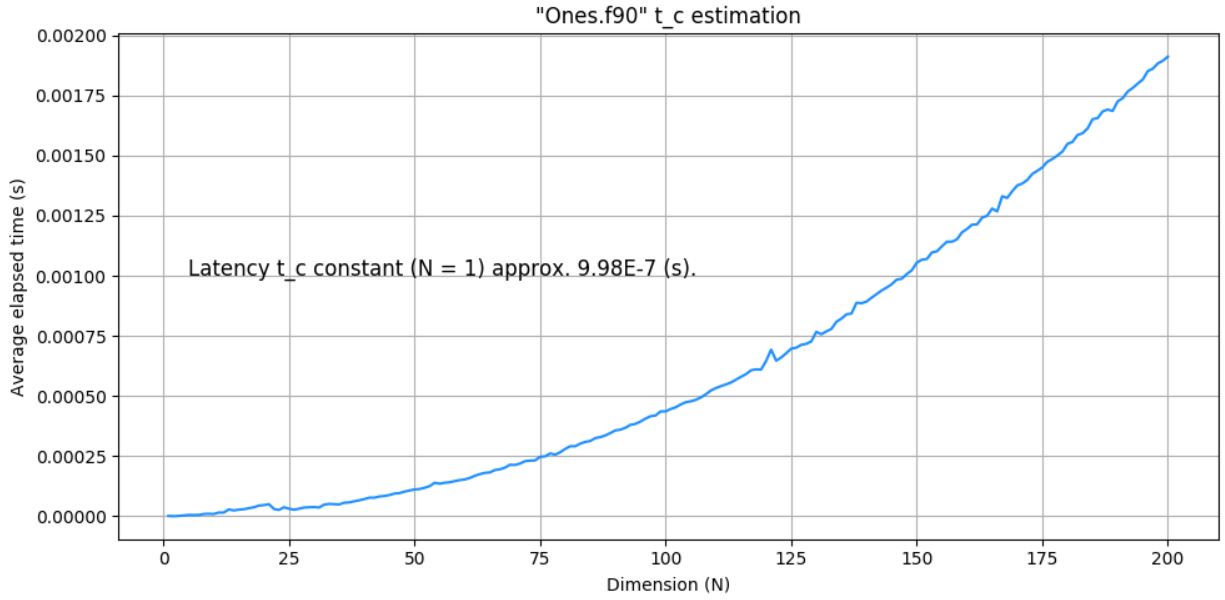

Figure 2: t_c estimation with `ones.f90` code. Code was adapted to operate on type "logical" rather than "int" (as I had in HW2).

Testing the model against the code execution, I obtained the following results

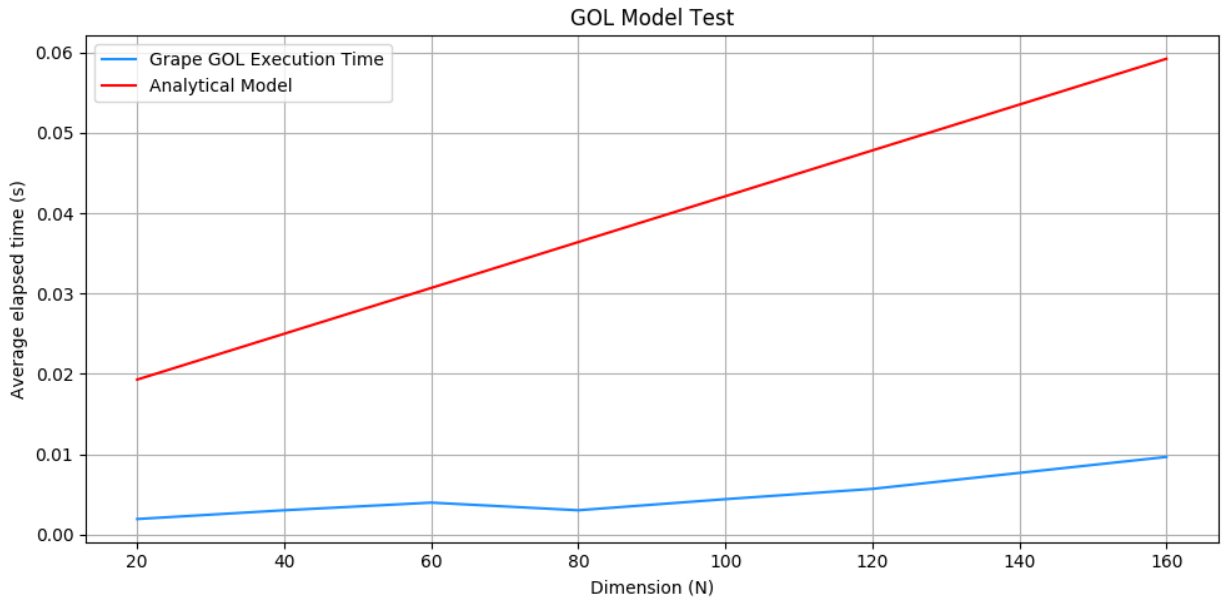

Figure 3: Analytical model vs actual GOL execution. Note number of elements in grid total to N^2.

The analytical model I constructed does not represent my obtained GOL execution times. This of course can be due to a misrepresentation in the model formulation, or even a miscalculated latency constant.

3 Code output

Here I include my GOL screen output at steps 0, 20, 40, and 80.

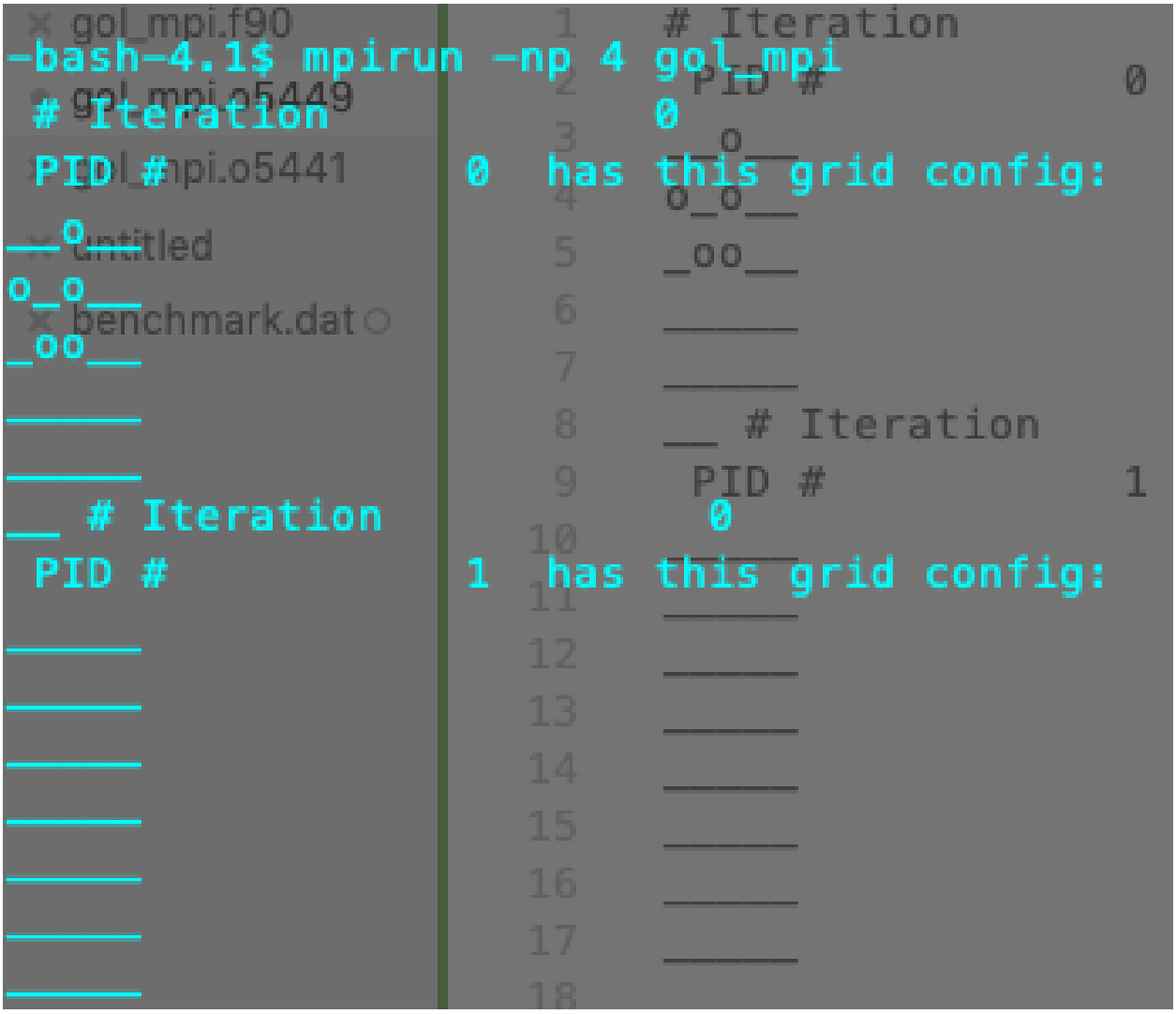

Figure 4: GOL at step 0. Terminal output was being a bit wacky, but here we see the glider on processor 0.

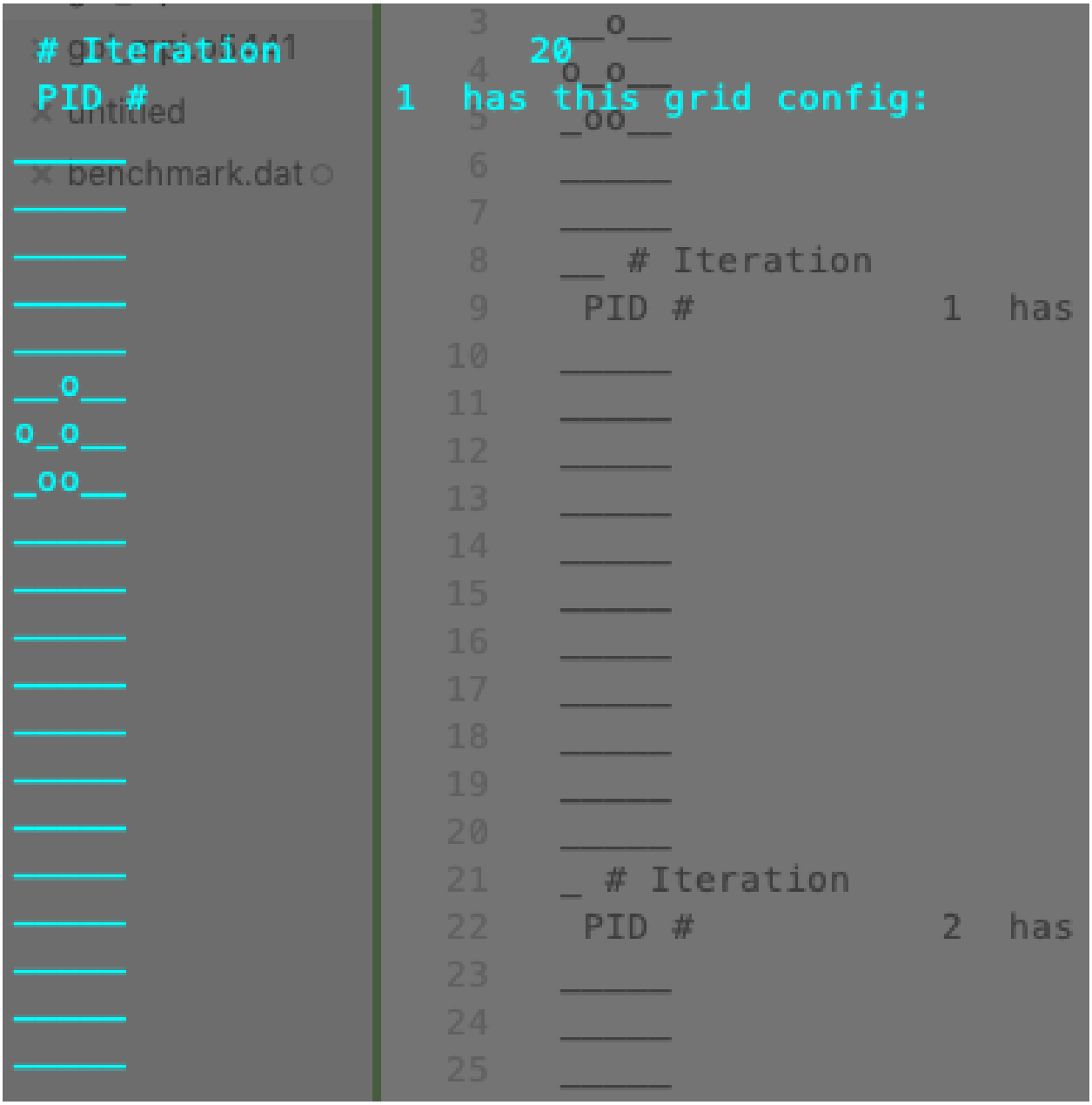

Figure 5: GOL at step 20. Processor 1 now has our glider. All others (not pictured) are empty configurations.

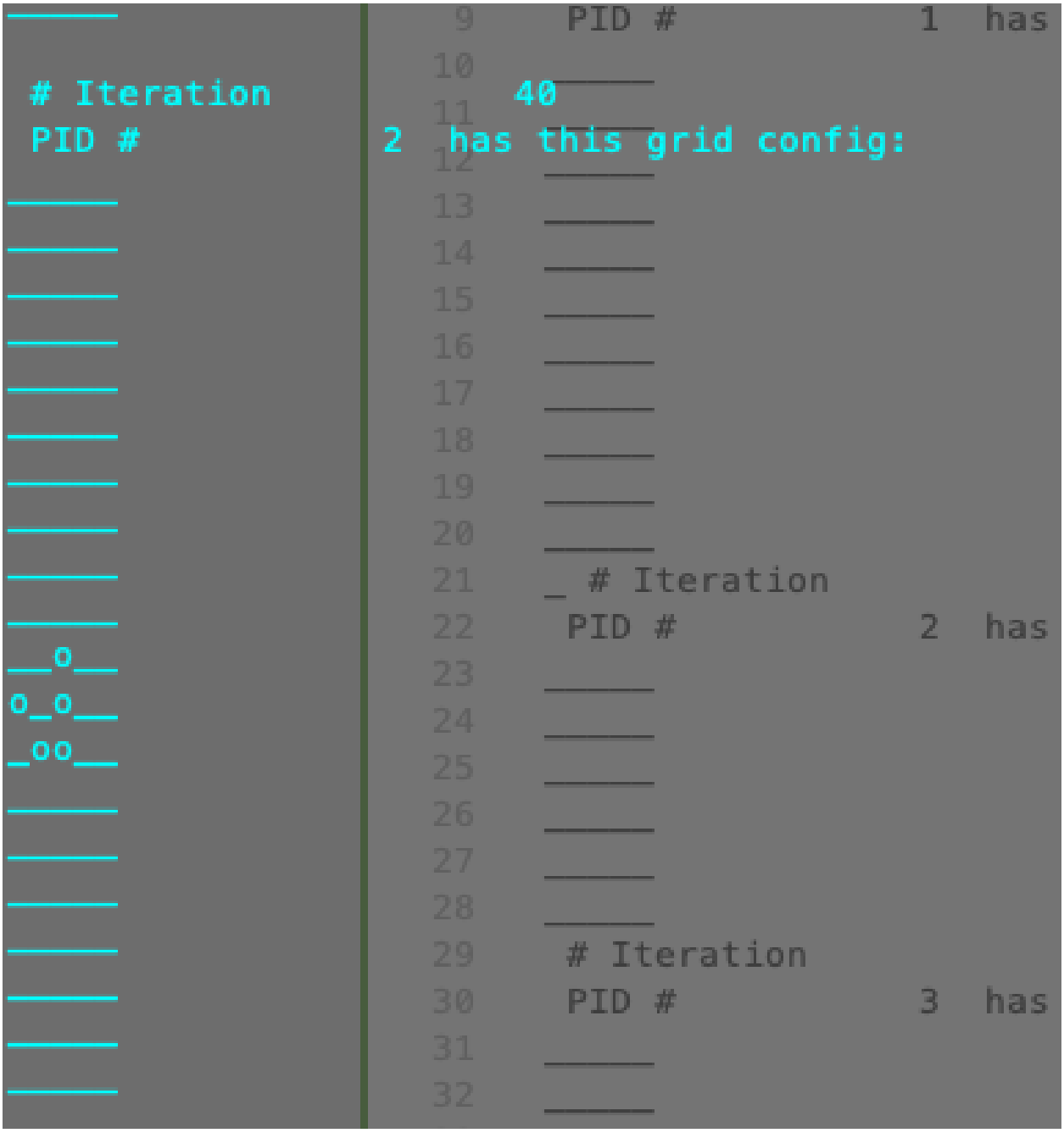

Figure 6: GOL at step 40. Processor 2 now has our glider. All others (not pictured) are empty configurations.

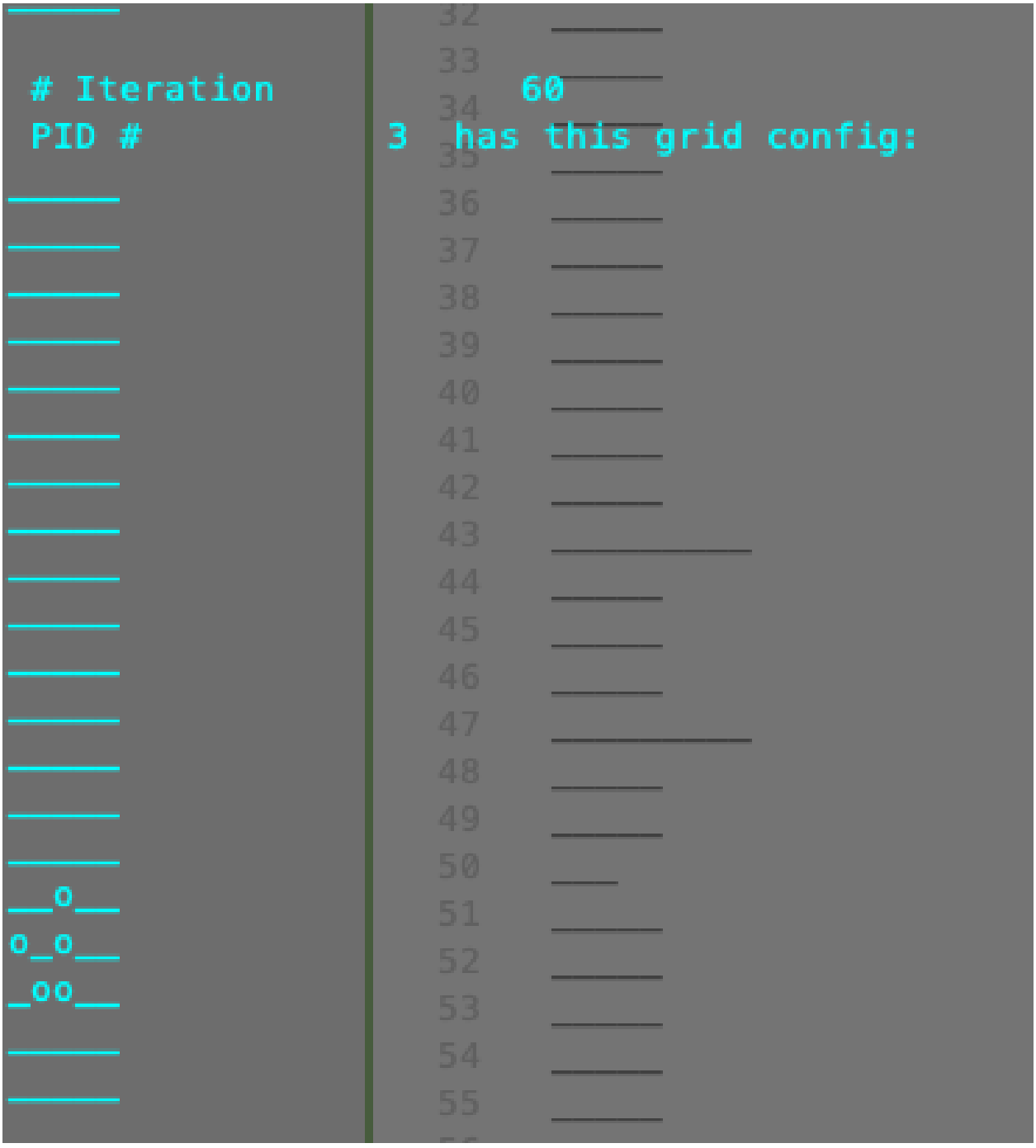

Figure 7: GOL at step 60. Processor 3 now has our glider. All others (not pictured) are empty configurations.

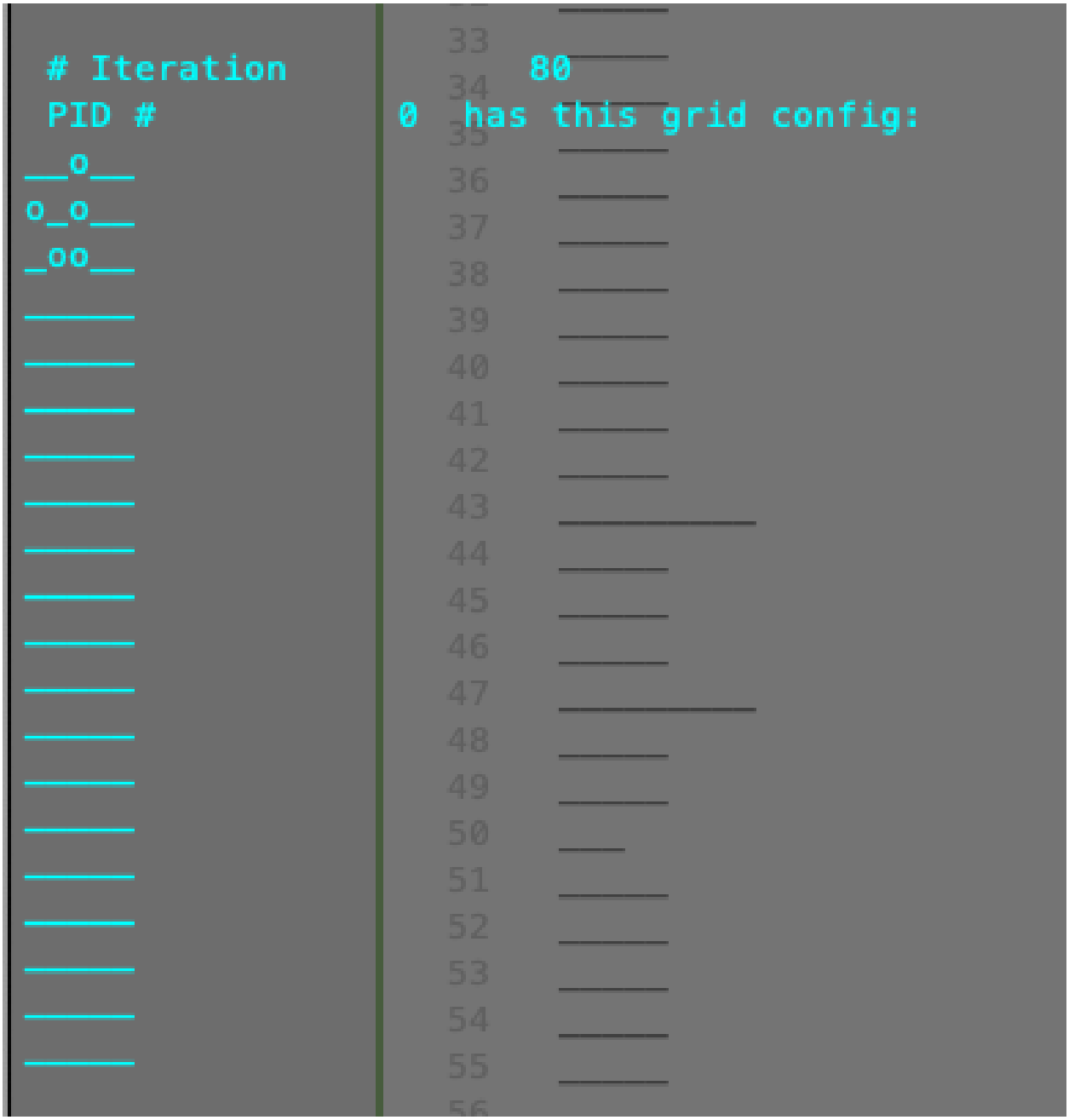

Figure 8: GOL at step 80. We've made it back to where we started on processor 0!