Gosper's Algorithm (Hashlife)

An explanation of how the hashlife algorithm reduces processing time and effort for cellular automata evaluation.

For my Master's dissertation I did some research using Gosper's Algorithm (also known as hashlife), which is an algorithm for hashing 2-dimensional cellular automata so that running the cellular automata is less memory and processor intensive. Anyway, I am told that the explaination I wrote in my disseration was reasonably comprehensive so I've decided to put that section of my dissertation online. Hopefully it is of use to any unfortunate students who need to try and understand how Hashlife works and don't want to spend the best part of a week stumbling through the originial paper; you can stumble through this as well.

If you need it, here are the references to Gosper's original paper and for my dissertation.

- Gosper, R. W. (1984). Exploiting Regularities in Large Cellular Spaces. *Physica D: Nonlinear Phenomena* , 10 (1-2), 75-80.
- Owen, Jennifer (2008). Complexity Measures on the Game of Life. *MSc Dissertation, University of York*.

Introduction

For this project we will be using Gosper's algorithm (Gosper, 1984) to simulate the game of life. It is highly efficient when compared to other simulators because it uses hashing and a mechanism Gosper named "macrocells".

Hashing

Hashing is a means of storing data structures in a table, called a hashtable, so that they are easily accessed later. The data structure is reduced to a number somehow and this number is used to reference where that data structure is stored in the hashtable. Different data may reduce to the same number, so the hashtable must create a linked list at that location to store all the different entries. Upon retrieval, hashing can significantly reduce the amount of time spent searching for a data structure.

Ordinarily to retrieve data the entire table must be searched each time, but using hashing only the data which reduces to the same number (known as a hash value) needs to be searched. With large quantities of data, such as could be created by a Game of Life simulator, this gives a significant reduction in search time.

Macrocells

A macrocell is a square of cellular automata 2^n by 2^n in size. Each macrocell stores pointers to 5 smaller 2^{n-1} by 2^{n-1} cells. Four of these cells are the four quarters of the larger macrocell, as shown in **Figure 1**, the other cell is what Gosper referred to as the "RESULT" of the larger macrocell. The RESULT square is concentric to the larger macrocell as shown in **Figure 2**. It contains the state of the macrocell after 2^{n-2} time steps, and it is entirely determined by the state of the macrocell. The reason that the RESULT macrocell is 2^{n-1} in size is that it will take 2^{n-2} time steps for information outside the macrocell to penetrate the RESULT cell's border. Hence, the RESULT square is the only information that can be accurately predicted from the initial macrocell after 2^{n-2} steps.

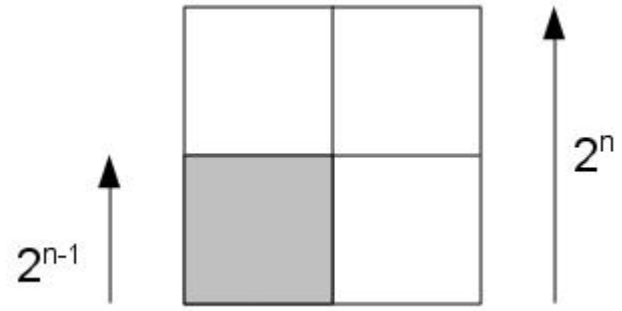


Figure 1: The macrocell is divided into 4 equal parts, each part is an individual but smaller macrocell.

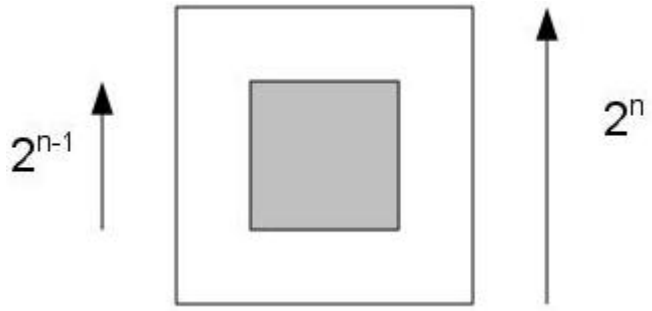


Figure 2: Macrocell showing the dimensions of itself and the RESULT square.

The most important thing to note about macrocells is that they are recursive. Each macrocell contains 5 smaller macrocells, these macrocells also contain 5 macrocells, and so on down to a single cell size 2^0 x 2^0; of which, for the Game of Life, there are only 2 (live and dead).

Each macrocell can also be reused. If a macrocell is a repeat of a previously discovered one then it will behave in exactly the same way, so it becomes a pointer to the previously discovered cell in storage. There is no need to store another copy of it in memory. Only if a new macrocell is found do we calculate its RESULT and save the macrocell into memory.

Calculating the RESULT

When a macrocell is 4 by 4 in size it is easy to calculate the RESULT using brute force, the cell only needs to advance one time step. However problems arise when n>2. If we take a macrocell and its 4 quadrants we can see that the RESULTs from the quadrants do not fill the area required to calculate the larger RESULT (**Figure 3**). Furthermore these RESULTs are from 2^{n-3} time steps into the future not 2^{n-2}, because the existing RESULTs are from the quadrant cells of size 2^{n-1}, and so their RESULT is at time $2^{(n-2)-1}$, or 2^{n-3}.

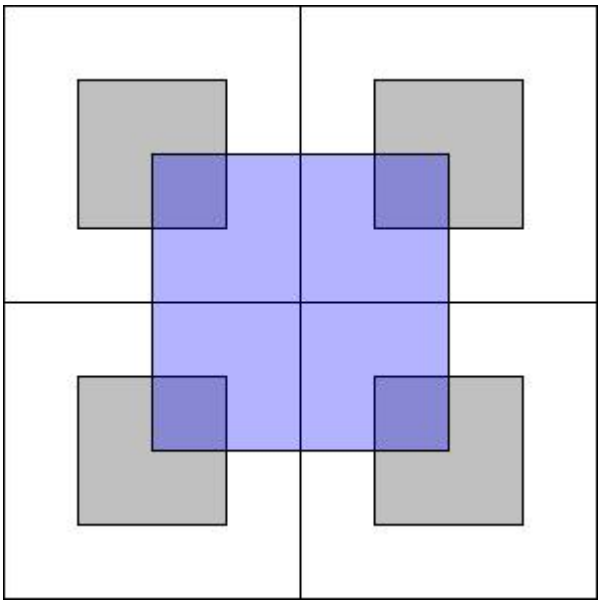
- 

Figure 3: The RESULTs from the quadrants are not sufficient to calculate the RESULT for the whole cell (shown in blue).

The solution Gosper proposed is scalable for all values of n, we will explain it here using an example where n=3.

The first step is to find the state of the macrocell after 2^{n-3} time steps, which in this example is a 6 x 6 square after 1 time step, the corners of which are already given by the RESULTs of the quadrants. The remaining area, indicated in **Figure 4**, is calculated by making five temporary 4 x 4 macrocells overlapping the original four, so that their RESULT regions end up occupying the blue area in **Figure 4**. This gives a 6x6 RESULT square which shows the state of the macrocell after 1 (2^{n-3}) time step.

The next step is much like the first. We need to reduce the 6 x 6 macrocell into a 4 x 4 macrocell that represents the 6x6 macrocell after another 2^{n-3} time steps (because $2^{n-3} + 2^{n-3} = 2^{n-2}$, which is the time in the future that we're interested in). The 6 x 6 square is converted to four overlapping 4 x 4 macrocells as illustrated by **Figure 5**.

The four 4 x 4 macrocells are used to evaluate their 2 x 2 RESULT, and the four 2 x 2 RESULTs are combined to make a 4 x 4 square. As the 6 x 6 RESULT is now being evaluated, the 4 x 4 macrocells are predicting another step into the future to give the RESULT after 2 time steps or 2^{n-2}.

For any value of n there are only two required stages for calculating the result: finding the RESULT after 2^{n-3} steps, and then using that to find the RESULT at 2^{n-2} steps. With larger values of n it is more likely we will encounter macrocells that do not have a RESULT; in which case these must be evaluated separately, starting at the smallest values of n and working upwards.

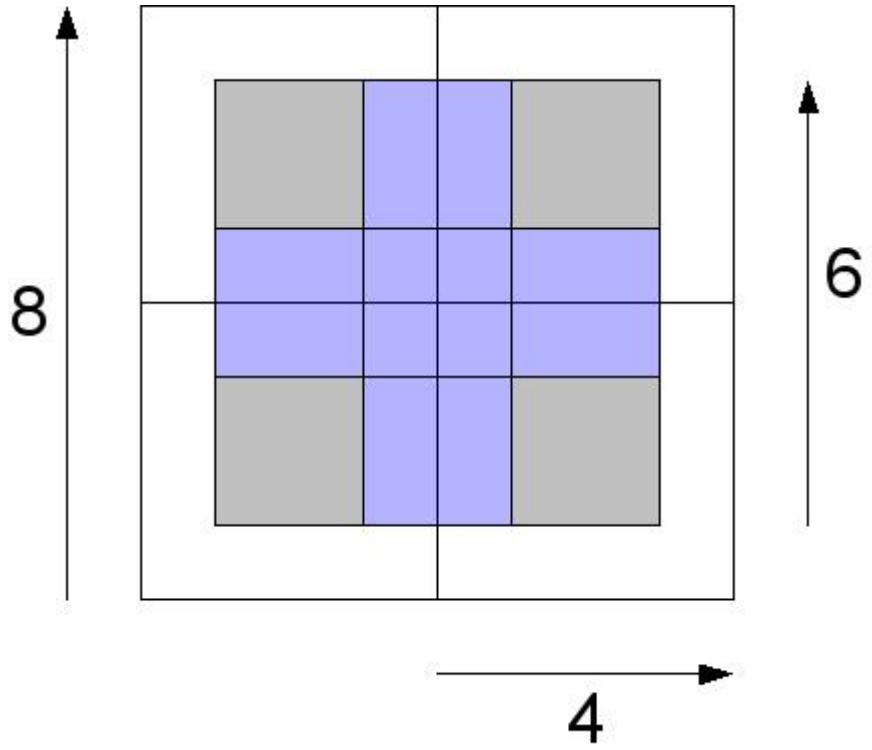


Figure 4: The area of the 6x6 square still left to find. The grey parts are known but the blue areas still need to be found.

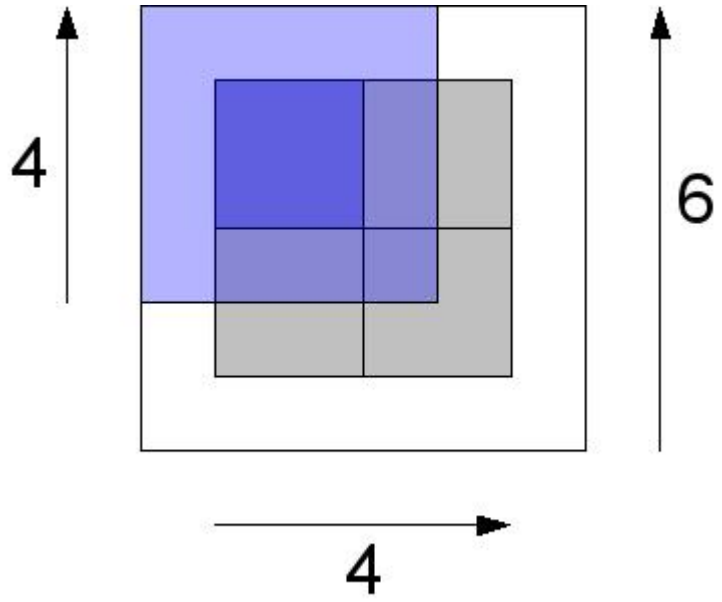


Figure 5: Reducing the 6 x 6 square to a 4 x 4 square. The blue area at the top left shows how a 4 x 4 macrocell is found in order to be reduced to a 2 x 2 RESULT. This is repeated to find the other three 2 x 2 squares.

Conclusion

The overall result of the algorithm is to consume much less memory than a naive alternative through the recycling of macrocells and the recursive nature of the algorithm. The macrocells are also able to predict their state 2^{n-2} steps into the future, and if n is large this also results in a significant loss of overhead.

Hashlife performs best for repetitive or periodic patterns as only a few macrocells are ever needed and they are reused often. For more chaotic patterns the algorithm is less efficient as macrocells are less likely to be repeated and so more are stored and the calculation of RESULTs uses more overhead.