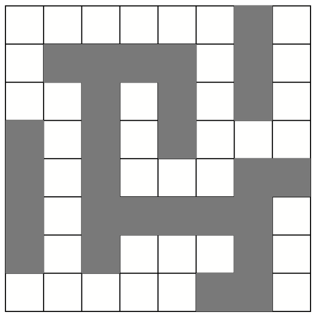
# Rethinking Maze Algorithms: Serial, Parallel, Power Constrained Parallel

Quentin F. Stout
qstout@umich.edu

This is a simple example to show why you often need to completely rethink algorithmic approaches for a parallel computer vs. a serial one, and a power-constrained parallel computer vs. a fully powered one. Each era introduces new models and reasons for studying them, and requires new approaches.

The example problem is to determine if it is possible to reach a goal position inside a maze, starting from the lower left corner. The maze is an $n \times n$ grid of squares, where each square is either white or black. Black squares are walls and white squares represent paths. You can move from a white square to an adjacent one that shares an edge, but not to one that only shares a corner. Here is a maze where you can go from the lower left to upper right, but not from the lower left to the lower right.



Algorithms will be described for three different scenarios:

1. The standard approach for a serial computer.
2. A very fundamental model of parallel computing, introduced by von Neumann in the 1950s. There is a grid of tiny processors (cores), each holding a square, where each is able to communicate only with its neighbors on the 4 sides and the 4 corners. They have only a fixed number of words of memory, independent of $n$, and they operate in a synchronized fashion. It is very easy to build such computers with myriad processors, all on a single chip. Many computers have been built based on a grid pattern of interconnections. They are sometimes called mesh-connected computers or meshes.
3. The same parallel model with the constraint that not all processors can be active at the same time.

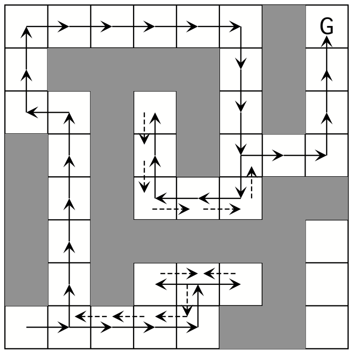
The power constrained model is new, and quite important. Chips now have so many circuits that it is impossible to have them all functioning at full speed without destroying the chip due to the heat. This "dark silicon" problem is worsening as transistor density increases, and it impacts systems from cellphones to supercomputers. This raises the question of whether we can design algorithms that are inherently efficient when only some of the cores are available, rather than automatically taking, say, twice as long if only half of the cores can be powered.

Another way to think of the power constrained model is to view it as small robots or other entities (electronic or biological) moving around on a surface, where the power represents the number of entities available.

## Simple Serial Algorithm

The standard serial algorithm is a simple depth-first search, moving from white square to an adjacent white square until either the goal G has been found or the search finishes without finding it. Here is an example of the ordering the squares might be visited, where at each square the white neighbors are visited in the order down, right, up, left. The dashed arrows represent backtracking. The algorithm takes $\Theta(n^2)$ time, and it is easy to show that this is optimal.

Note that the algorithm can also be viewed as the motion of an entity crawling along the surface.



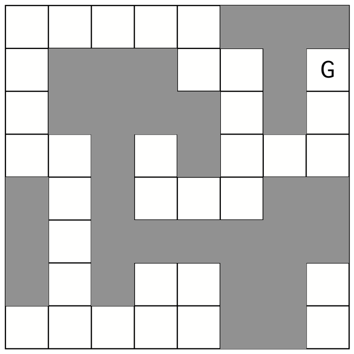
## Elegant Parallel Algorithm

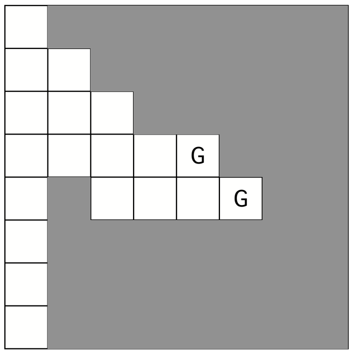
The serial algorithm is simple, but parallelizing it isn't since no one knows how to parallelize depth-first search. Depth-first search is P-complete in the NC $\neq$ P conjecture, which is basically a statement that not all problems with serial polynomial algorithms can be parallelized very well ("NC" is "Nick's Class", which you can google for a precise definition). It is similar to the P $\neq$ NP conjecture.

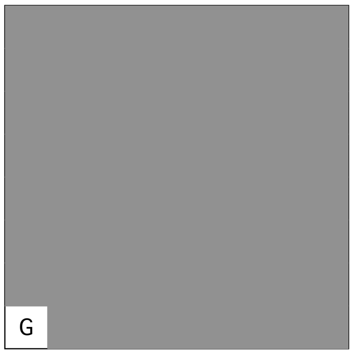
However, depth-first search is not necessary, and long ago Beyer [1] and Levialdi [2] gave a parallel algorithm for this problem, using the model described above. Their algorithm is so simple that it can be performed by a cellular automaton, i.e., the processors are copies of the same finite state machine, independent of $n$. The algorithm is similar to Conway's Game of Life.

At each step, a square which is white stays white except when its right and above neighbors are black, in which case it turns black. A square which is black stays black except when its right, above, and upper-right corner neighbors are white, in which case it turns white. Further, whenever a white square stays white, and shared an edge with (or was itself) a square marked G, then it becomes marked with a G. It is assumed that there is a border of black squares around the maze.

These snapshots show the configuration after the first step, an intermediate step, and the final one.







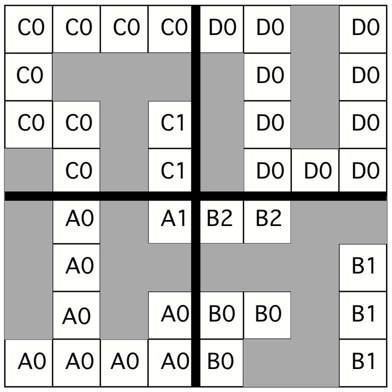
Beyer and Levialdi showed that this simple procedure solves the maze problem, in that the lower left corner becomes marked with a G if and only if the goal can be reached, and that this occurs in $2n-1$ or fewer steps. Thus the algorithm takes $\Theta(n)$ time, which is optimal if we take into account the fact that information separated by distance $\Omega(n)$ must be combined. Here distance is for any 2-dimensional surface: electrons in a chip, robots moving around, etc.

**Power-constrained Parallel Algorithm**

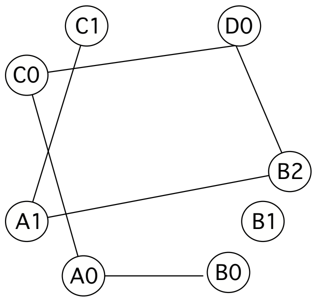
While the parallel algorithm optimizes time, it uses the standard approach of having all of the processors on all of the time. Hence if it takes unit energy to move information unit distance in unit time, the total energy is $\Theta(n^3)$. The serial algorithm takes only $\Theta(n^2)$ time, which is an approximation of the energy needed to run it, so a natural question is whether the problem can be solved with $\Theta(n^2)$ energy while still taking only $\Theta(n)$ time. Such an algorithm must have most of its processors off at any given time, or, equivalently, relatively few entities moving at any given time. The number of entities, or the maximum number of processors on at the same time, is the peak power. Despite the fact that energy and power are dominant concerns in computer systems, few parallel algorithms have been developed to minimize these resources.

A completely different algorithm will be used to reduce the power. It's based on graph component labeling and a divide-and-conquer approach. The components are the connected regions of white squares, and hence the goal square G can be reached if and only if the lower left square is in the same component.

Suppose the components in each quadrant have been labeled, as in the following figure. Notice that A0 and A1 are in the same component in the overall maze, but this cannot be determined by just examining the squares in the quadrant.



To determine the components of the complete maze one needs only consider squares along the edges of the quadrants. Pairs of adjacent white squares in different quadrants indicate that components should be merged. For example, A1 and B2 are in the same component since they adjacent. These pairs give the information in the following undirected graph:



Component labeling in the complete maze can be completed by determining the connected components of this graph. An important aspect of the graph is that it involves $O(n)$ edges, rather than $\Theta(n^2)$ squares. If the edge information is moved to the center then it fits into an $\sqrt{n} \times \sqrt{n}$ subsquare. There is a $\Theta(\sqrt{n})$ time parallel algorithm that can be used be used to determine the connected components, where the algorithm uses all of the subsquare processors all of the time. However, the total energy is only $\Theta(n^{3/2})$, which is less than the energy needed to move the edges to the center ($\Theta(n)$ words of information moving $\Theta(n)$ distance, using $\Theta(n^2)$ total energy). The movement can be done in $\Theta(n)$ time using peak power $\Theta(n)$.

The overall recursive algorithm takes $\Theta(n \log n)$ time using peak power $\Theta(n)$, for $\Theta(n^2 \log n)$ total energy (see [3] for details). It's an open question as to whether the maze problem can be solved in $\Theta(n)$ time using peak power $\Theta(n)$.

As I mentioned at the beginning, the purpose of this example is to show that there are significant differences between the standard serial algorithms that people learn versus parallel algorithms, and yet further modifications are needed to minimize energy. Optimizing the time of parallel algorithms is still an important research area, and optimizing time/energy tradeoffs of parallel algorithms is a research area that has just barely begun. For almost any algorithmic problem (geometry, graph theory, matrix multiplication, etc.) the time/energy tradeoffs are open questions, and additional factors, such as computing on a surface vs. in 3 dimensions, generate additional questions. People are developing 3-d chip modules to minimize time and energy, which is one of the reasons that algorithms for computing in 3 dimensions are becoming of interest.

It will also be interesting to measure the energy and/or time actually used on various parallel computers, including GPUs (graphics processing units) and other accelerators.

If you're interested in working on problems in this area please contact me: qstout@umich.edu

1. Beyer, W.T., *Recognition of Topological Invariants by Iterative Arrays*, Ph.D. thesis, M.I.T., 1969.

2. Levialdi, S., "On shrinking binary picture patterns", *Commun. ACM* 15 (1972), 7–10.

3. Stout, Q.F., "Algorithms minimizing peak energy on mesh-connected systems",
   `http://web.eecs.umich.edu/~qstout/abs/SPAA06power.html`