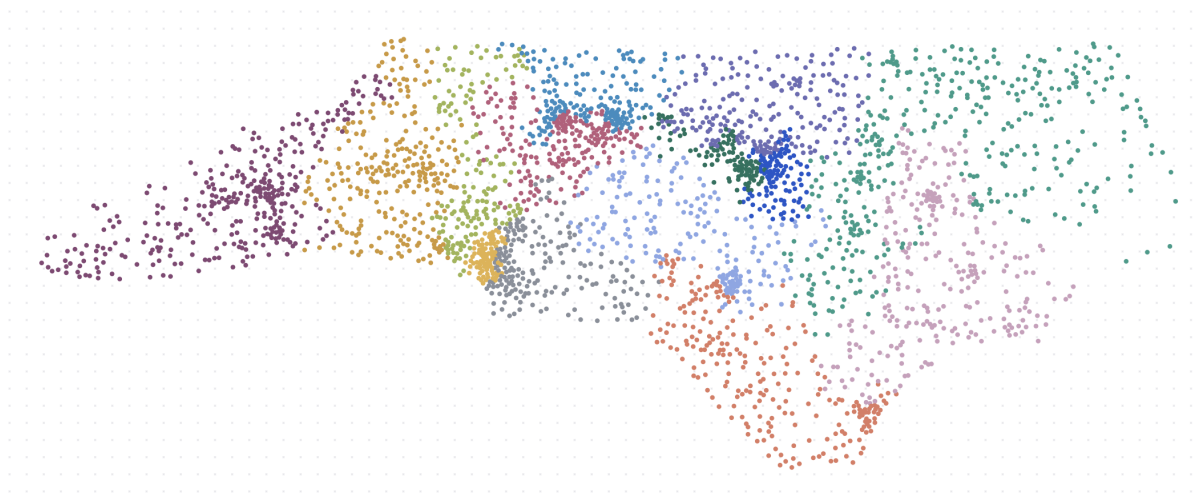


Every Map Everywhere All at Once

Good Algorithms Finish First: GPU-Accelerated Neutral Mapping for Modern Sampling

Ryan McComb · October 2026 · mccomb.ca/writing/everymap



One neutral map out of 10,000, drawn in 2.0×10^{-4} seconds each.

“Every rejection, every disappointment has led you here, to this moment.”

Waymond Wang, *Everything Everywhere All at Once* (2022)

Many have we had, redistricting fights, over the past few cycles, and the evergreen topic has only come to more of a point recently. However, for all the statistical firepower that gets thrown at these cases, most of them hinge on something far less exotic than fixing aggregation bias in EI:¹ a comparison between the map someone actually drew and the maps nobody in particular drew, with *almost all* cases being just more complicated derivatives of that one question.

But redistricting maps are hard to make, and humans don’t really have the bandwidth to make millions of them to see the natural distribution of maps, so, as we do for almost everything, we’ve created computer programs to do such menial work.² The answer the academics and courts have settled on is wonderfully brute force: draw a lot of maps. Not a handful of alternatives by hand, but thousands of them, en masse, created by a random process that follows

¹Though that has been done too. I seem to reference MSOA in every article I write, so I might as well get it out of the way: VoteHub’s MSOA methods paper was all I read for a few weeks about a year ago.

²Even though each US appeals court could employ a subsection of ET that just makes maps on Redistricter and DRA all day... food for thought.

the legal rules and never once looks at a single vote. That pile of maps, an "ensemble" of neutral maps, tells you what the geography does on its own,³ and then you can see where a real (gerrymandered or not) map will fall onto this god-like distribution.⁴

The catch is that this is slow! Immensely so, and with the minutia of an attention span that we possess as of current,⁵ it seems fitting that a faster way to sample is a necessity to be created. Ten thousand effective neutral maps of North Carolina take *redist* about four minutes on my ten-core MacBook Air, and a big state can take a whole lot longer. So, as one does, I built my own sampler, which I'm calling Shoal,⁶ that draws the exact same kind of maps, from the exact same statistical target, in about two seconds on the same laptop! That is roughly a hundred times faster, and this post is about how it works, why it is still exact, and why good algorithms finish first!

Divinely Inspired Maps

Take North Carolina, which elects fourteen of its own Tarheels to Congress each cycle. Chop its 2,666 voting precincts into fourteen districts of equal population, each in one connected piece, cut up at (basically) random. Count how many districts Kamala Harris would have carried in 2024.⁷ Then do it again. And again, and then like a few thousand more times! Press play below and watch the pile build.

³My theory is that while God was resting on his seventh day he logged into his Redistricter Pro account and created natural geographical districts, and I think these programs are the best way to distill God's incredible work.

⁴The gold standard here is *redist*, an R package from Harvard's ALARM Project built on the SMC method of McCartan and Imai (2023). Ensembles like these have shown up as evidence in redistricting cases in a bunch of states.

⁵It seems to me that human attention went down as we started discovering attention for large language models, but I digress.

⁶Like a shoal of fish: thousands of them, all moving at once. You get it, niche GPU reference, or not that niche maybe.

⁷Or weighted by expected seats, even though it's more complicated, un peu.

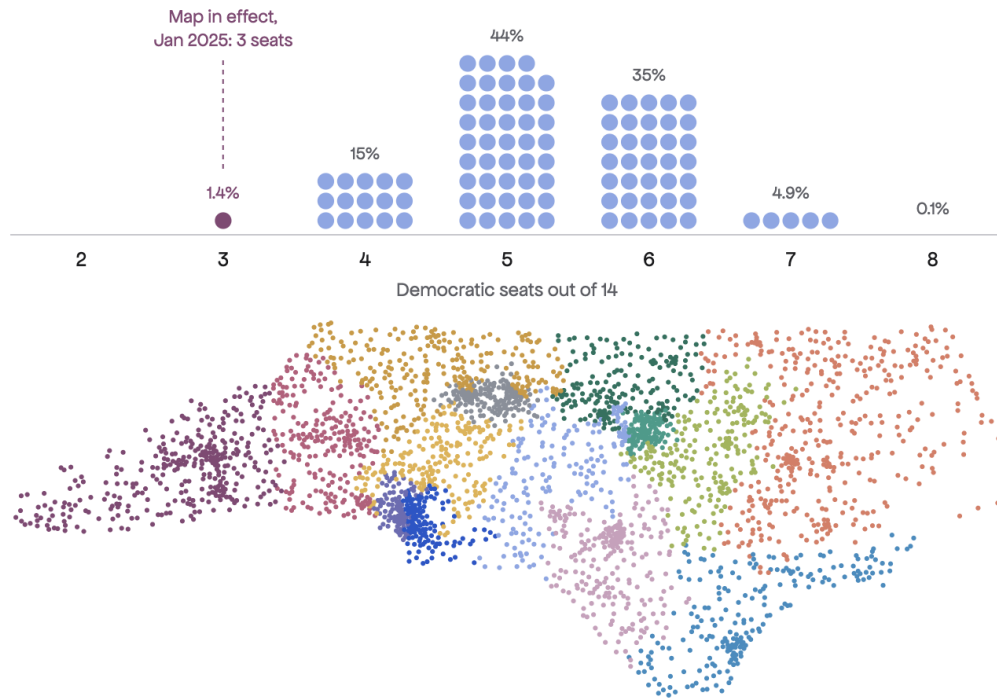


Figure 1: Each dot is 100 neutral maps, stacked by how many seats Democrats would win under 2024 results. The map in effect in January 2025 gives them three, fewer than about 98.5% of the neutral maps!

That divinely inspired distribution is broadly the whole idea: it's what North Carolina's geography produces on its own, naturally. And due to factors inherent to mapping and districting, a statewide vote of 48.4% doesn't mean a fair map hands Democrats 48.4% of the seats.⁸ Voters aren't spread out evenly; Democrats are packed into cities, which wastes their votes in these neutral maps almost by definition. The neutral maps bake all of these nuances in automatically,⁹ and what they say is that this particular state, with this particular geography, usually gives five seats, and sometimes four or six. Where a real map lands in that pile is often key evidence in a redistricting case; what it means is manifold, and certainly something far above my pay grade of making free, detailed blogs on a niche redistricting concept on my personal website.¹⁰

⁸Though if you are creative enough, it could hand you way more than 48.4%.

⁹How convenient!

¹⁰These maps use only equal population and contiguity. Real rules usually add more, like keeping counties whole, which changes the ensemble. The sampler does that too; I left it out here to keep the comparison with *redist* apples to apples.

Trees, Trees, Trees, a Forest?

The core trick is older than any of this newfangled GPU magic. To split a region into two pieces of equal population, draw a random spanning tree of its precincts (a set of connections linking every precinct with no loops) and cut a single edge somewhere. Snip one edge of a tree and you always get exactly two connected pieces, so the only question is whether some edge leaves them with equal population.¹¹ If one does, congrats, you have a split. If none does, you draw another tree.

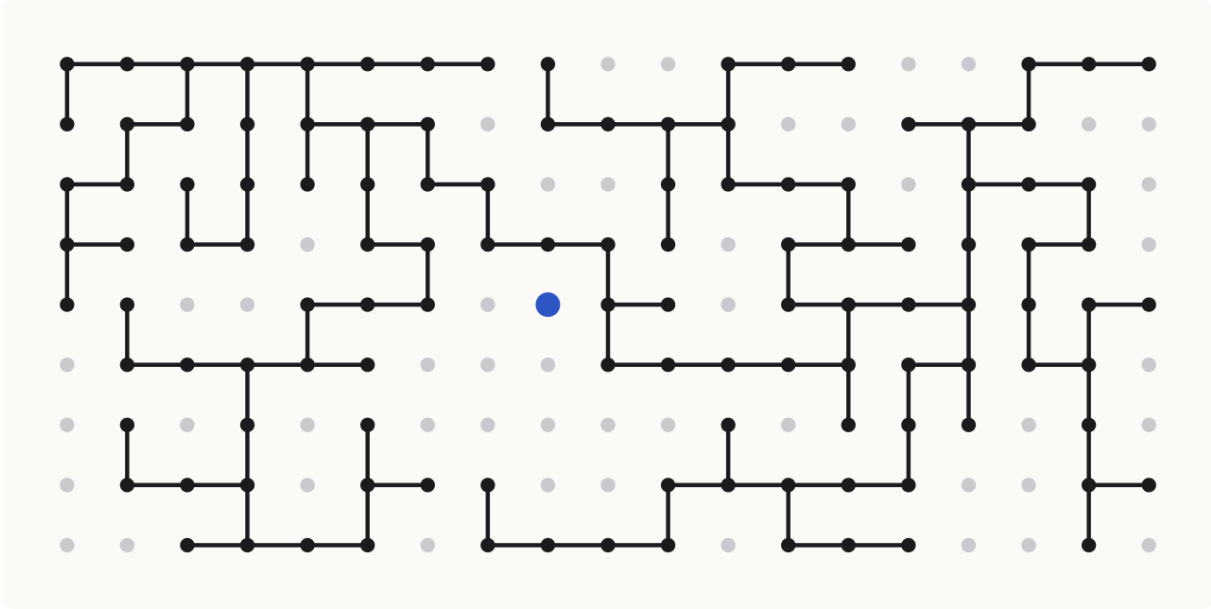


Figure 2: A uniform random spanning tree, drawn with Wilson’s algorithm: a random walk (blue) wanders until it bumps into the tree, erasing any loop it makes along the way, and the path it leaves becomes a new branch. Once everything is connected, the tree is cut at an edge that splits it in half!

To draw a whole map, you just repeat this, carving one district off the remaining region at a time, fourteen times for North Carolina. Done naively, though, this cheats: maps that happen to be easy to carve show up way too often. The fix to this problem is to give every map a weight that corrects for exactly how likely the process was to spit it out, which is where the statistics come in.¹²

¹¹Roughly equal, in our case. Real congressional maps are held to near-exact equality (the Supreme Court says “as nearly as practicable,” and states usually get within a handful of people), but neutral maps are built from whole precincts, which can’t hit that, so they allow a small tolerance: every map in this post is within $\pm 0.5\%$ of equal.

¹²The quality of this article is about to decrease exponentially, as my statistics knowledge is not particularly good: it is just what I’ve gleaned from watching... and learning from Wikipedia and other math online.

Necessary Casualties in a Random Walk through the Forest

Okay, time for the one fancy equation in this post. You don't need to understand it to follow along, but I think it looks cool,¹³ it's the key to this whole thing, and if I can understand it, you can definitely learn it:¹⁴

$$\pi(\xi) \propto \prod_{i=1}^k \tau(\xi_i)$$

Read left to right, it says: the chance of drawing any particular map, $\pi(\xi)$, is proportional to (that's the $\propto$) the number of spanning trees inside each of its k districts, $\tau(\xi_i)$, all multiplied together (that's the big $\prod$). A compact, chunky district can be wired up in an enormous number of ways, so it has a gigantic number of spanning trees; a long, stringy district has far fewer possibilities. That means maps made of compact districts come up far more often, which is exactly what you want from a neutral map, and unlike using ReCom and annealing,¹⁵ we don't have to specify flags for the things we want (compactness!).

Sequential Monte Carlo (SMC) builds thousands of maps side by side, one district at a time. After each round, every half-finished map carries a given weight corresponding to how over- or under-represented it is in the overall ensemble, and then the whole population gets resampled again: the half-finished maps the target favours get copied and keep advancing through the process, and the ones it does not get axed.¹⁶ At the end, the weighted maps are an exact sample from the target. Survival of the fittest, so to speak, but for congressional districts. Fun stuff!

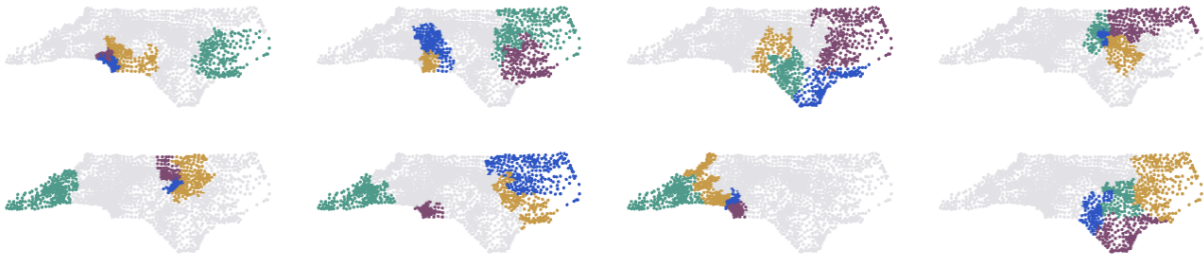


Figure 3: Eight maps of North Carolina built in parallel, one district per GPU step. Obviously, the sampler runs thousands at a time, but I don't think you want to watch a thousand little dots on your screen.

¹³The pi symbol is cool! I first got introduced to it on my pre-calculus final a year back, in a problem our teacher hadn't taught us how to do, since the only way through is a telescoping sum:

$$\sum_{n=1}^{999} \log \left(\sqrt[3]{\frac{n^2}{n^2 + 2n + 1}} \right)$$

And fun fact: a sum of logs is secretly the log of a product, so this Σ is a $\prod$ in disguise.

¹⁴This is the spanning forest distribution with compactness parameter 1, the default in *redist*, and the target both samplers are measured against in this post.

¹⁵ReCom, short for recombination, is the Markov chain from DeFord, Duchin and Solomon's "Recombination": it merges two neighbouring districts and redraws them with a random tree. Paired with simulated annealing, as in Mosaic, it becomes an optimizer, and there you have to set weights for everything you want, compactness included.

¹⁶A sad but necessary casualty of the process.

Because the maps are built (mostly) independently, this is exactly the kind of work a GPU is made for.¹⁷ As you may or may not know, a GPU has thousands of small cores that all run the same short program at the same time, and “draw a tree, find a cut” for one half-finished map is a very short program. This type of algorithm has usually been run on a CPU, which has only a handful of cores and is built for doing things one after another, very quickly. That’s not really what this job needs; it needs thousands of the same small thing done all at once.

Out of Pure Benevolence

Here is the fun part. Below, Shoal, my new GPU-accelerated sampler, and *redist* both race to 10,000 effective neutral maps of North Carolina, in real time, at the speeds they actually run on my laptop.¹⁸ Each square is one map, coloured and shaded by how many seats Democrats would be favoured to win in it.

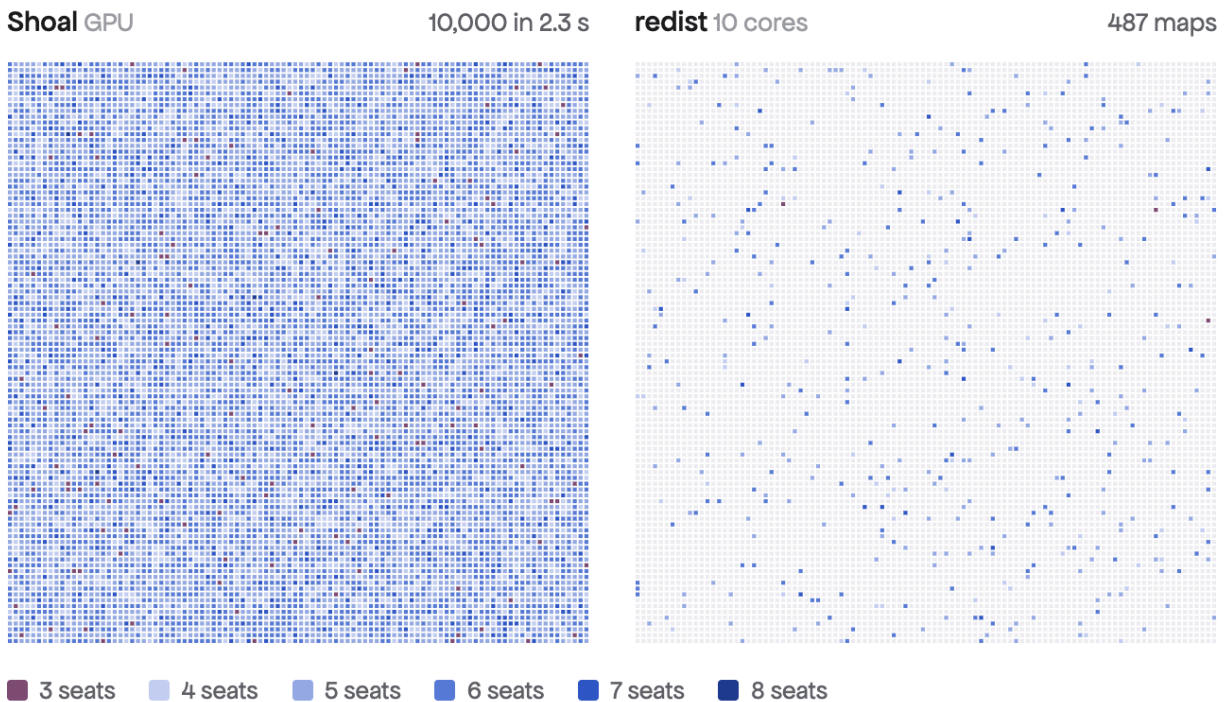


Figure 4: A real-time race to 10,000 effective neutral maps of North Carolina. Shoal finishes in about 2.3 seconds; *redist*, at its measured velocity, would need about four minutes.

Yes, *redist* does eventually finish, in about four minutes, though based on how slow it is, you would have no inclination to ever think it would. Out of pure benevolence, I didn’t want to make you wait.

All told, that works out to about a hundredfold gap (4,417 effective maps a second against 40.6,

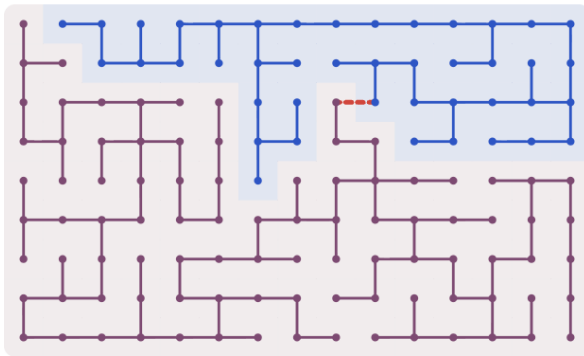
¹⁷Jensen Huang would be proud.

¹⁸An Apple M5 MacBook Air, probably the second-best investment I’ve ever made, other than McKesson stock or some really good election trades I’ve made on Manifold.

or roughly 109 times faster), and it comes from two key optimization ideas, only one of which is about the new hardware we are utilizing for these computations.

The first is how many maps actually count, and this is the part I think is genuinely clever on my part.¹⁹ Remember the weights from earlier, the fix for the cheating, where maps that are easy to carve would otherwise show up way too often? Every time a sampler carves a district off, it has to give the new partial map one of those weights, correcting for how likely the process was to produce it. The obvious way is to ask how likely the exact cut you made was. The trouble here is that the very same split can come out of the process in a variety of disparate ways, so a weight built and tuned from a single path is noisy, inefficient and frankly dumb: two identical partial maps can end up with wildly different weights just because of happenstance, namely which edge happened to get cut. More a map of circumstance than one of design, and certainly not a map we want!

One path: the cut that was made



Every path: all cuts that make this split

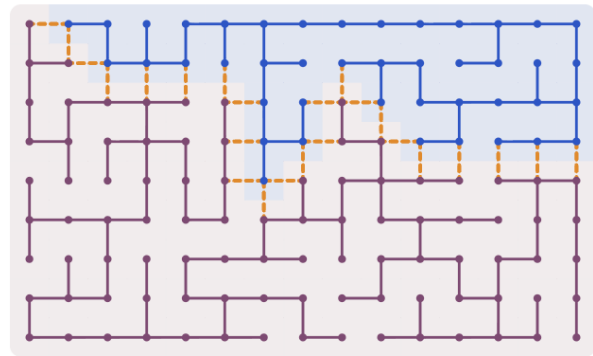


Figure 5: One carved district, weighted two ways. Left: the usual weight looks only at the one edge that was cut (red). Right: the optimal backward kernel counts every edge along the new border (orange), since cutting any of them could have produced this exact split.

The optimal backward kernel asks an objectively better question: not “how likely was the path I took?”²⁰ but a notably different one: “how likely was it to end up here by any path at all?” The kernel sums over all the edges along the new border that could have produced an identical split, so all that path-to-path luck smooths out and the weights come out far less lopsided and “wonky”.²¹ In this weighted sampling, a map with a teeny weight contributes almost nothing, which is intuitive, and what matters far more to the whole process is the effective sample size, roughly how many equally weighted maps the whole pile is worth, in the context of the broader set. About 83% of this sampler’s maps count, against about 45% for *redist*, so it throws away roughly half as much work before any clever engineering at all, which means every actual engineering breakthrough on top of it gets multiplied in how useful it is later in

¹⁹The optimal backward kernel itself isn’t mine: it comes from Pierre Del Moral, Arnaud Doucet and Ajay Jasra’s 2006 paper, “Sequential Monte Carlo samplers”. Applying it to redistricting was my idea, though, and it’s actually what inspired this whole project; I got it during a swim practice one morning! I only ported it over to the GPU once I also figured out that would be an improvement, and ideating with Opus 5.5, Fable 5.1 and Astra 6 was invaluable along the way.

²⁰“I” being the random walk. Wow, we’re really anthropomorphizing computer programs now.

²¹It’s almost like adding DEI to mapping algorithms: it understands more holistically what you’ve gone through in life.

the post.²²

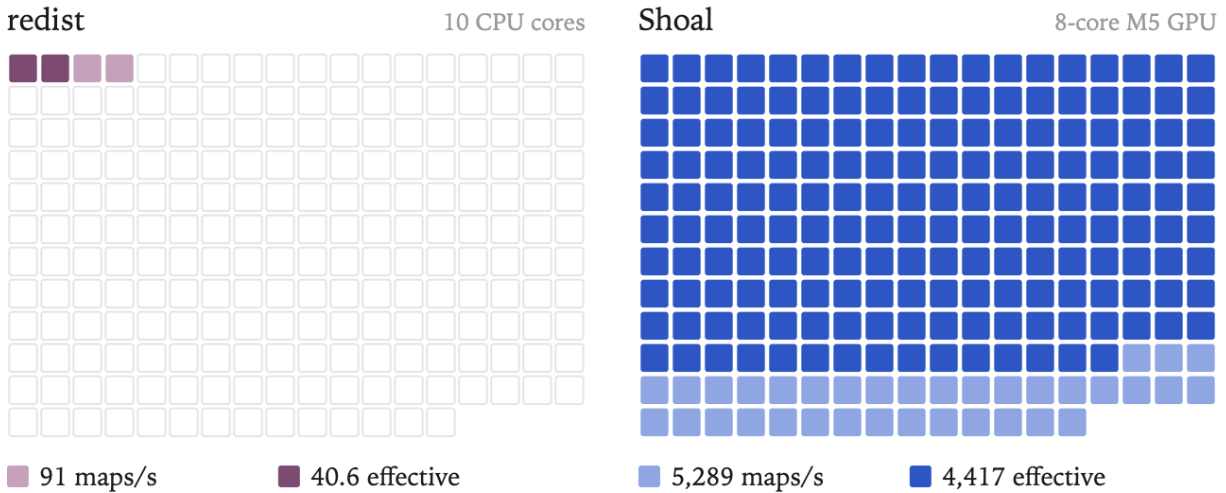


Figure 6: Neutral maps of North Carolina per second on the same MacBook Air. Each square is 25 maps; the dark ones are the effective ones.

The second part is raw speed, and the GPU is most of it, but certainly not all! On the CPU alone, no GPU, the same sampler is 17 to 29 times faster than *redist* across three states, and almost ten times faster on a single core. *redist*’s core is already compiled C++, so this is not a “Python slow, Rust fast” story;²³ it is careful engineering for one very specific job, plus the weights above. The GPU then stacks another three to six times on top.²⁴

Telemetry Data

Everything here was measured on one machine, an Apple M5 MacBook Air (ten CPU cores, eight-core GPU), with both tools sampling the same target from the same precinct graph. I think the following graphs are pretty self-explanatory.²⁵

²²Yes, I am foreshadowing!

²³As much as I, and a lot of other people who like Rust, wanted it to be.

²⁴“Effective” here is Kish’s effective sample size, $(\Sigma w)^2 / \Sigma w^2$, of the final weights. Both samplers resample between rounds, so their maps share ancestors and the honest count of truly independent maps is somewhat lower, for both of them.

²⁵I feel very victorious as I write this; I’m currently playing “We Are the Champions” by Queen.

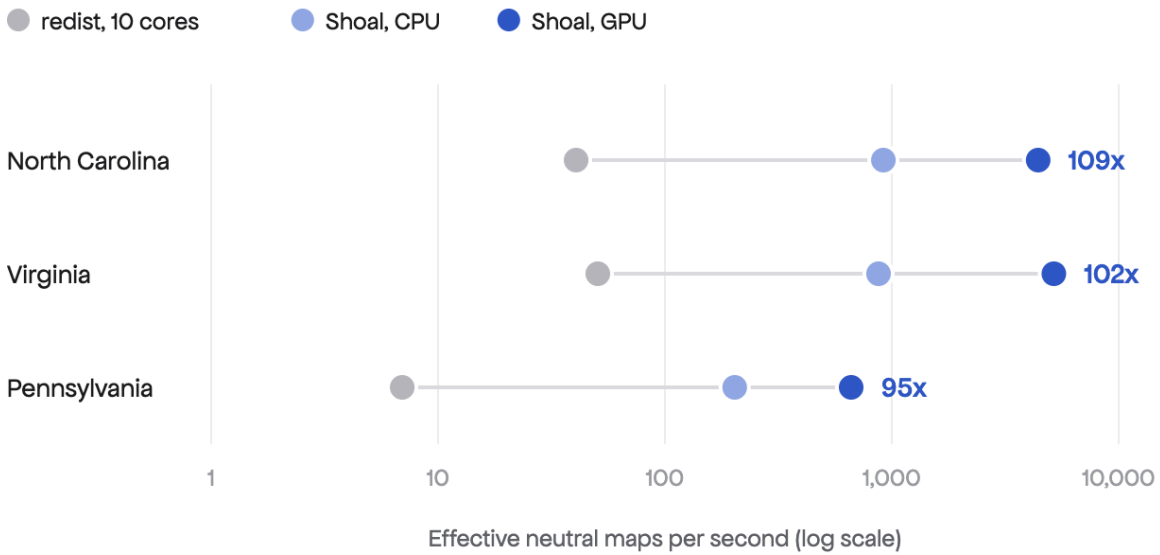


Figure 7: Effective neutral maps per second, log scale. The GPU sampler is 95 to 109 times faster than *redist* on the same laptop; the CPU version alone is 17 to 29 times faster.

	<u>redist</u> 10 cores	CPU 10 cores	GPU 8-core M5	GPU vs <u>redist</u>
North Carolina 2,666 precincts, 14 districts	40.6	918	4,417	109x
Virginia 2,465 precincts, 11 districts	50.6	874	5,179	102x
Pennsylvania 9,178 precincts, 17 districts	6.95	203	662	95x

Effective maps per second, medians.

In practical terms, running an ensemble used to mean “start the job, go get lunch, and hope it didn’t crash your computer, leak memory or fail in a million other ways, and that it did it correctly.” Now it finishes while you are still looking at the screen, and I believe that changes what you can actually do with it. If you change a rule, add a county constraint or nudge a tolerance, you can see the new range of neutral maps right away, and run it as many times as you like (for fun?).²⁶

²⁶Settings for every number here: population within 0.5% of equal, compactness 1, no county rule; *redist* 4.3.2 with *redist_smc* defaults on all cores. Medians of three to five runs. Pennsylvania used 2,000 maps per run because

No Mo(r)E

”Speed” is easy if you are allowed to be wrong,²⁷ so the quality that is of paramount importance is whether the GPU samples exactly the right distribution of maps; anything less than basically being spot on is a massive failure of the program. Running this check on a real state is essentially impossible, because there are absurdly many possible maps.²⁸ However, on a tiny grid it is not an impossible task: a five-by-five grid split into five districts has exactly 4,006 valid maps, no more and no less, few enough to list every single one and compute its exact probability. Then you can check whether the samplers land on those exact numbers, an efficient way of checking my new sampler’s homework.

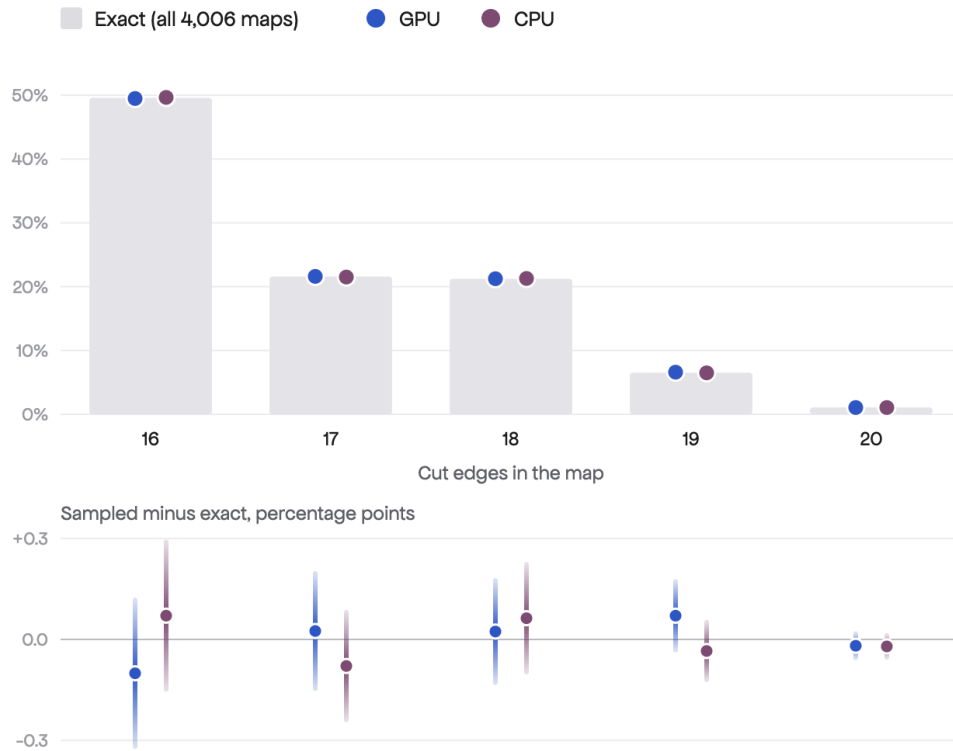


Figure 8: On a five-by-five grid, the exact share of maps with each number of cut edges (bars) against what the GPU and CPU samplers estimate over 300 runs each (dots). Below, the difference from exact, with two-standard-error bars: every one covers zero.

To my initial surprise, both land on the exact answer within ordinary sampling noise. On real states, where listing every map is hopeless, every map the GPU produces can also be re-checked on the CPU, tree by tree and weight by weight; in tests on North Carolina and Texas, more

redist needs a couple of minutes for each, and I am impatient.

²⁷I learned this as a kid, when I would blow through standardized tests very fast and make stupid mistakes.

²⁸Nobody knows exactly how many. A five-by-five grid has 4,006 ways to make five equal districts, a seven-by-seven grid already has about 159 million, and the count explodes from there. With 2,666 precincts, North Carolina almost certainly has more possible maps than there are ways to shuffle a deck of cards, about 8×10^{67} , which is itself far more than the number of grains of sand on Earth.

than 160,000 of those checks agreed to about twelve digits!²⁹

Spare a Few Million?

The sampler lives inside Tessera, a set of algorithmic redistricting tools I’ve built and will launch in some given fashion in the future. It also complements an optimizer that draws maps toward whatever goals you set. Having both cool algos in one place, with the neutral side fast enough to run on a whim, is really the broadest point: you can draw your own map, ask how unusual it is and how much it sticks out from the “average” pile, change a parameter, and ask again, all in the time it used to take to start one run. For anyone who draws, studies or argues about maps,³⁰ that’s of immense use.

Huge thanks to Cory McCartan and Kosuke Imai, whose work on sequential Monte Carlo for redistricting and *redist* this builds on directly, to the MGGG Redistricting Lab for ReCom, and to Matt Mohn for Mosaic. Map data comes from the U.S. Census Bureau, the Voting and Election Science Team, the ALARM Project and Dave’s Redistricting.³¹

And if you’re wondering where this goes next, look at Bitcoin mining. It went from CPUs to GPUs, then to FPGAs, then to ASICs, chips designed to do exactly one thing. Redistricting just made it to GPUs, and honestly, don’t kill me for saying this, but the job is quite ASIC-shaped: tiny random walks and integer adds over a graph small enough to fit entirely on a chip.³² So the redistricting ASIC is left as an exercise for the reader (or for me, if anyone has a spare few million dollars lying around). Until then, maybe I’ll try building an FPGA version sometime.

There is still plenty to do! I have county rules running on the GPU in testing (a twenty-thousand-map ensemble of Texas with counties kept whole takes under a minute), and the optimizer is next on the list. More on both soon.

Citation

McComb, Ryan. “Every Map Everywhere All at Once: Good Algorithms Finish First.” *mccomb.ca*, October 2026. <https://mccomb.ca/writing/everymap/>

²⁹I really did do my due diligence on this, I swear.

³⁰Election Twitter, and pretty much no one else.

³¹I need to give some props to Claude here: a lot of the code was written by Opus. I was more the person who came up with the idea and the implementation, wrote some pseudocode, and then mostly gave it a lot of time and nodded along as it wrote Metal shaders, which I could not tell you the first thing about.

³²North Carolina’s whole precinct graph, ~7,500 connections, fits in a few hundred kilobytes, and the GPU’s bottleneck is fetching it from memory, which my laptop has plenty of, but which isn’t the fastest thing to reach. This is exactly what a custom chip with the graph on board would skip. A real tape-out costs millions, of which I don’t have even a millionth to spare, so an FPGA would be the sane first step! If anybody wants to help make one with me, I would love to!