Nick's Class: Do many hands make light work?

Kartik Chandra, kach@cs

CS 254B, Spring 2019-20

Contents

1 The classes $\mathbf{NC}_k$

On the first day of CS254B, the professor thought he had walked into the wrong lecture hall. It was standing room only in Hewlett 200 — never before had a complexity theory class at *any* university seen such enrollment numbers. But the professor wasn't mistaken; it really was the golden age of complexity theory classes at Stanford. As he shuffled his notes around on the table waiting for 9:30AM he couldn't help but try to count the number of students. "It looks like there are n of you," he said, "Welcome to CS254B!"

Meanwhile, head TA Nick's heart rate was going up rapidly. As he looked out at the crowd, he could only think one thought: "That's a *lot* of grading." He did not look forward to doing $\Theta(n)$ work every week. With conference deadlines coming up, he knew he could only reasonably spend $O(\text{polylog}(n))$ time on his part-time TA-ship.

Then Nick had an idea. He opened up his laptop and began an email to the department chair:

> Dear chair:
>
> Would it be possible to allocate funding for extra CS254B TAs?

Soon enough, he got a response:

> Sure. How many do you need? Would $O(1)$ be enough?

Nick thought for a moment. $O(1)$ TAs would still have to do $\Theta(n)$ work each. That was no real improvement, was it? Nick decided to push his luck.

> That might not be enough. Could we spring for poly(n)?

And soon enough came the response:

Done. We'll hire some co-terms over the weekend.

Nick breathed a sigh of relief and with all his heart thanked the various entities that fund undergraduate education in the CS department. At least this first assignment would be just $O(1)$ work for him...

From here on out, we will say that problems which Nick's team of $O(\text{poly}(n))$ TAs can solve in $O(\log^k(n))$ time are in Nick's Class $\mathbf{NC}_k$ and we'll define the union of all these sets of problems as $\mathbf{NC} = \bigcup_{i=1}^{\infty} \mathbf{NC}_i$. To be clear,

$$\mathbf{NC}_1 \subseteq \mathbf{NC}_2 \subseteq \cdots \subseteq \mathbf{NC}_i \subseteq \cdots \subseteq \mathbf{NC}$$

In a sense, we can regard problems in $\mathbf{NC}$ as "efficiently parallelizable," especially in a world where you can rent thousands of CPU cores for just a few dollars.

2 Why $\mathsf{SORT} \in \mathbf{NC}_1$

The first problem set was all graded and ready to be handed back, but before that the teaching team wanted some statistics on how students did.

"Would you be able to sort the n psets by grade so that I can get a sense of the distribution?" Nick asked Sara, one of the TAs.

Sara shrugged. "That's $O(n \log n)$ work, and grades are due in just a couple of hours," she said.

"That's true, but maybe we can split up the work among the teaching team?"

Sara thought for a moment. "Sounds hard."

"Why?" Nick was on a roll. "Suppose we did mergesort in parallel. We could divide up the assignments, have different parts of the team sort those chunks simultaneously, and then merge them real quick."

" 'Real quick?' Merging at the top level is still $O(n)$ work, Nick."

"Oh," said Nick, and he fell silent, thinking.

"But I like where you're going. How about this: each of us takes a paper and places it where it belongs in order. That sounds like $O(1)$ work for each of us, right?" Sara cleared off n tables in the basement room of Gates that the TAs were using as headquarters. Then she put the tables in a row, and labeled them $1, 2, \ldots, n$ to illustrate her point. "So the best paper goes on table $n-1$, and the worst paper goes on table 0, and..."

$$\begin{bmatrix} A^+ & C^- & B \end{bmatrix} \to \begin{bmatrix} 2 & 0 & 1 \end{bmatrix} \to \begin{bmatrix} C^- & B & A^+ \end{bmatrix}$$

"Sure, Sara, but how do I know where my assignment belongs? Its rank depends on everyone else's papers."

"You're right. But maybe the pairwise comparisons can happen simultaneously."

"I see!" Nick got up and arranged n^2 tables in a grid. "Suppose we get n^2 of us to consider each possible pair of papers and decide which one has a higher grade." Nick began putting post-it notes with a 0 or a 1 onto the desks to make his argument clear.

$$\begin{bmatrix} A^+ & C^- & B \end{bmatrix} \to \left[\begin{array}{c|c|c} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{array}\right] \to \begin{bmatrix} 2 & 0 & 1 \end{bmatrix}$$

"...oh, but counting is still an $O(n)$ problem" said Nick, disappointed again, "add this to the list of failed ideas."

"That's it!" said Sara, "Adding! Instead of *counting* the 1's, let's *add* them."

"Isn't that the same thing?" said Nick.

"Not quite," said Sara, "Because adding a long list of numbers is just like your mergesort but the merging is super fast."

$$\overbrace{\overbrace{1+0}^{\text{Added by Alice at } t=1} \quad + \quad \overbrace{1+1}^{\text{Added by Bob at } t=1}}^{\text{Added by Alice at } t=2}$$

"Yes!" said Nick, somewhat redeemed, "We can do that in $\log(n)$ time, if you consider each addition to be $O(1)$."

"Close enough," said Sara.

So they got to it. After an easy $O(1)$ minutes spent grading all n psets, the poly(n) TAs broke off into n teams of n each that compared their assigned psets pairwise, doing n^2 comparisons in another $O(1)$ time. Finally, each team added their comparison bits hierarchically in $O(\log n)$ time and rearranged the assignments based on the resulting rankings. This was the hardest part, and they were done, all things considered, in $O(\log n)$ time. The students got their grades back on Tuesday night, just in time for the Homework #2 to go out on Wednesday...

3 Why $\mathbf{NC}_1 \subseteq \mathbf{L}$

This arrangement worked quite well for a couple of weeks. But then, on the Monday of week 4, Nick walked into the basement of Gates and found almost all of the poly(n) tables gone. In fact, there were only $\log(n)$ tables left.

"DAHA tables? EOM" he asked the course staff mailing list ("Does Anyone Have Any tables? End Of Message").

"Oh, we took them for the CS254B midterm," wrote Sara, "We had to hold it out on the football field because no lecture hall could accommodate all n students."

"Huh," said Nick, "How are we going to sort the papers today? There isn't enough space to work."

"I've been thinking about that," said Sara. "You know how until now we've been able to get grading done in $O(\log(n))$ time? That means each of us could only ever possibly look at what was on $O(\log(n))$ tables."

Nick nodded, wondering where this was going.

"So what if we took turns doing what we *used* to do simultaneously?"

"What do you mean?"

"We can share space. First TA #1 goes and does whatever she *would* have done, using the $O(\log(n))$ tables however she wants. Then TA #2, and so on..."

"Aha! So we fit our whole process into $O(\log(n))$ tables. Excellent!" Nick was getting excited. "But how long will it take if nothing can happen in parallel?"

Sara sighed. "It'll take some time, but there's no new problem set this week..."

"Sounds good to me," said Nick, "let's do it." His crew went to work. First, the TAs arranged themselves in a DAG, based on whose tasks they depended on to do their own tasks. Then, Nick called out names in a depth-first-search order; each TA when called did his or her task in $O(\log(n))$ space, leaving behind the result for the next TA, and erasing anything unnecessary. By the end of the week the homeworks were all graded and sorted.

Though they may not have realized it at the time, Nick and friends had shown that any problem that they could solve in $O(\log^{k=1}(n))$ time (i.e. $\mathbf{NC}_1$) was in $\mathbf{L}$, the class of logspace problems. Hence, $\mathbf{NC}_1 \subseteq \mathbf{L}$.

By the way, there is a hidden assumption here that the "structure" of the computation is easy for Nick to work out in order to coordinate the TAs. To be more formal about this we would have to define a *uniform* $\mathbf{NC}_1$.

4 Why $\mathbf{NL} \subseteq \mathbf{NC}_2$

It was the morning of the final, and the campus-wide Internet was down thanks to a CS144 assignment gone rogue. This would be fine — the exams were already printed — but the professor had not yet told students where the final would be and now couldn't use Piazza to communicate with his students.

"Do we have *any* other way to contact students?" he asked Nick.

"Not everyone," said Nick. "I only have Andrea's phone number because I was her RA in FroSoCo last year."

"I think Zoë's on the undergraduate class council, she might be able to mass-text everyone," said Sara, "Can Andrea reach Zoë?"

"I don't know," said Nick, "but I bet she can reach Barry. Does Zoë know Barry?"

"No," said Sara, "But Yan knows Zoë, they did psets together. Maybe Barry knows Yan?"

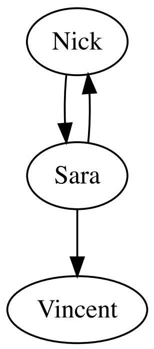
"Hmm," said Nick, "I'm not sure they know each other. But let's not give up, maybe we can find another path from Andrea to Zoë. We're computer scientists, we know how to check graph connectivity."

Sara sighed, "That sounds like a lot of time we *don't* have. DFS is poly(n) work."

"But there are poly(n) of us!" said Nick.

"One of these days saying that is not going to save you," said Sara, "How are we splitting up the work *this* time?"

Nick took a moment to think. This was a tough one. "Let's write down who we know is able to reach whom, first." He picked up the sharpie. "Here's a small example. Suppose we're looking for a path from myself to Vincent."



"This is getting unwieldy already," said Sara, "Maybe it's better to write this information in a nice, clean grid instead."

$$\begin{bmatrix} N \rightarrow N = 1 & N \rightarrow S = 1 & N \rightarrow V = 0 \\ S \rightarrow N = 1 & S \rightarrow S = 1 & S \rightarrow V = 1 \\ V \rightarrow N = 0 & V \rightarrow S = 0 & V \rightarrow V = 1 \end{bmatrix}$$

"You mean a matrix?" said Nick.

"Sure, a matrix." Sara erased the labels. "That starred entry is zero, which means there's no direct one-step connection from Nick to Vincent."

$$G = \begin{bmatrix} 1 & 1 & \boxed{0^*} \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

"What about an indirect connection — say, at most *two* steps away?" asked Nick.

"That's easy, we can just do casework on who the intermediate person would be," said Sara:

$$G_{N \to_2 V} = \bigvee_{X \in \{N,S,V\}} (G_{N \to X} \wedge G_{X \to V})$$

"This looks awfully like matrix multiplication," said Nick. "I wonder..."

$$G^2 = \begin{bmatrix} 2 & 2 & \boxed{1} \\ 2 & 2 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

"Isn't this the *number of ways* to go from N to V in two steps?"

"You're right," said Sara, "Actually, I think in general $(G^p)_{N \to V}$ is represents the number of ways to go from N to V in p steps."

"Why?" said Nick.

"Induction," said Sara mysteriously.

"Okay, okay. " said Nick. "Well, we'd only ever need up to n steps, because there are only n students. I suppose this means that checking whether there is any path is simply checking whether $(G^n)_{N \to V} = 0$."

"That's not too bad," said Sara. "We can exponentiate matrices with just $\log(n)$ matrix multiplications by repeatedly squaring them."

"How fast can we multiply matrices?"

"I think we can have n^2 subteams of TAs do each cell in parallel," said Sara, "And each cell is just an n-sized addition problem which we already know how to solve in log-time using the hierarchical summation trick we used when sorting."

"Okay, altogether that's $O(n^3)$ TAs and $O(\log(n) \cdot \log(n)) = O(\log^{k=2}(n))$ time... yes, we can do this!"

So Nick and friends organized themselves into n^2 teams of n, each dedicated to a cell of the Stanford CS theory social graph matrix. They repeatedly squared this matrix (using $O(\log(n))$ time per squaring) a total of $O(\log(n))$ times, and discovered that Andrea could indeed reach Zoë. A quick game of "telephone" later, all n students were assembled on the Oval ready to take the CS254B final.

It turns out that checking connectivity between vertices in a directed graph is a *complete* problem for the class **NL**, which represent problems that can be solved in logarithmic space by a nondeterministic Turing machine. (Why? Given a log-space Turing machine, take each of the polynomially-many possible states and treat it as a vertex in a graph. Set up edges corresponding to transitions, and ask "is there a path from initialization to accept?") With their algorithm, Nick and friends have shown that $\mathbf{NL} \subseteq \mathbf{NC}_2$.

5 Epilogue: Is NC = P?

Nick and his teaching team received *glowing* reviews that quarter. The next fall, Nick and Sara went on to become the CS 161 head TAs, helping teach introductory algorithms. For the first assignment, students were asked to implement bubblesort, and run it on the size-n class roster to alphabetize the names.

“Okay, team, you know the system — let’s get started!” said Nick when it was time to start grading.

Sara paused. “I’m not sure we can grade these assignments in $O(1)$ time each. What if the student’s program takes $\Omega(2^n)$ time? We could be in here until Sunday waiting for it to end — if it ends at all! What if it doesn’t even *halt?*”

Nick frowned. “You’re right. If it takes longer than $O(\text{poly}(n))$ time,” he said, “We should just write ‘TIMEOUT’ and move on.”

“That’s still $\text{poly}(n)$ work per person, though.”

“But there are $\text{poly}(n)$ of us, surely we can parallelize as we always have?” Nick got to work trying to devise a scheme to get it done in $O(\log^k(n))$ time with his $\text{poly}(n)$ friends.

...and he’s still working, to this day! If Nick’s TAs could solve this problem, then they could efficiently solve any problem in **P** by writing a program to solve it in $\text{poly}(n)$ time and then accelerating its execution across the $\text{poly}(n)$ workers. On the other hand, it’s possible that there are problems in **P** that Nick’s team just cannot solve in polylog time — these problems would be *inherently sequential*. Over time, Nick and friends have found many examples of *complete* examples for this class of inherently-sequential tasks; they call these problems **P-complete**.

To summarize their results so far, however:

$$\mathbf{NC}_1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}_2 \subseteq \mathbf{NC} \subseteq \mathbf{P}$$

For all we know, each of these inclusions could be an equality...

My main source for this paper was *Limits to Parallel Computation: P-Completeness Theory* by Greenlaw, Hoover and Ruzzo. You can find it online for free here: `https://homes.cs.washington.edu/~ruzzo/papers/limits.pdf`

In the real world, Nick’s Class is named after computer scientist Nick Pippenger for his work on circuits that are large but not too deep, which correspond to efficiently-parallelizable programs. (For this reason, in textbooks such as Arora-Barak you will typically find **NC** discussed in the context of circuits.)