

September 23, 2026

Today: Start parallel algorithms.

Template for parallel algos in computer algebra:

- Express problem as solving system of linear equations
- Invoke parallel solvers off-the-shelf

Why parallelism?

- Interesting questions & answers
- Not all problems that admit efficient algos have parallel algos. Fine-grained measure of complexity.

Machine model:

- $\text{poly}(n)$ processors
- shared memory (no comm. cost)
- arithmetic op ($+$, $-$, $\times$, $\div$) in unit time.

Goal: linear algebra in $O(\log^2 n)$ parallel time

- determinant
- matrix inverse
- solve $Ax = b$
- characteristic polynomial

Classic algo: Gaussian elimination
Efficient, but highly sequential.

Idea: find a different representation
of the objects we want to compute

Recall: for $A \in \mathbb{C}^{n \times n}$,

$$\begin{aligned} P_A(t) &:= \det(tI - A) \\ &= \prod_{i=1}^n (t - \lambda_i) \end{aligned}$$

is characteristic polynomial of A

($\lambda_1, \dots, \lambda_n \in \mathbb{C}$ eigenvalues of A)

$$p_A(t) = t^n + c_{n-1} t^{n-1} + \dots + c_1 t + c_0$$

$$c_1 = -(\lambda_1 + \dots + \lambda_n) = -\text{trace}(A)$$

$$c_n = (-1)^n \lambda_1 \dots \lambda_n = (-1)^n \det(A)$$

Obs. $p_A(\lambda_i) = \lambda_i^n + c_{n-1} \lambda_i^{n-1} + \dots = 0$

$$\sum_i p_A(\lambda_i) = (\lambda_1^n + \dots + \lambda_n^n) + c_{n-1} (\lambda_1^{n-1} + \dots + \lambda_n^{n-1}) + \dots + c_1 (\lambda_1 + \dots + \lambda_n) + n \cdot c_0 = 0$$

Write $s_i := \lambda_1^i + \dots + \lambda_n^i$. Rewrite $\uparrow$ as

$$s_n + c_{n-1} s_{n-1} + \dots + c_1 s_1 = -n \cdot c_0$$

Allows computing c_0 from $s_1, \dots, s_n, c_1, \dots, c_{n-1}$

Similarly, $s_{n-1} + c_{n-1} s_{n-2} + \dots + c_2 s_1 = (-n-1) c_1$

$\uparrow$ Set $\lambda_n = 0$, divide $p_A(t)$ by t , eval at $\lambda_1, \dots, \lambda_{n-1}$

Lemma (Newton-Girard Identities)

$$\text{let } S_i := \lambda_1^i + \dots + \lambda_n^i$$

$$\begin{pmatrix} -1 & & & & \\ S_1 & -2 & & & \\ S_2 & S_3 & -3 & & \\ \vdots & \vdots & \ddots & \ddots & \\ S_{n-2} & S_{n-3} & \dots & -(n-1) & \\ S_{n-1} & S_{n-2} & \dots & S_1 & -n \end{pmatrix} \begin{pmatrix} c_{n-1} \\ c_{n-2} \\ \vdots \\ c_1 \\ c_0 \end{pmatrix} = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_{n-1} \\ S_n \end{pmatrix}$$

This is useful because

- 1) $S_i = \text{trace}(A^i)$ can be computed quickly in parallel.
- 2) Triangular matrices can be inverted in parallel.

Lemma $\text{trace}(A), \text{trace}(A^2), \dots, \text{trace}(A^n)$ can be computed in $O(\log^2 n)$ parallel time.

Proof Recall $\text{trace}(A) = \sum_{i=1}^n a_{i,i}$.

Suffices to compute A^d . Use repeated squaring.

$T(n, d)$ = parallel time to compute A^d
when A is $n \times n$.

$$A^d = A^{\lfloor \frac{d}{2} \rfloor} \cdot A^{\lceil \frac{d}{2} \rceil}$$

$$T(n, d) \leq \max(T(n, \lfloor \frac{d}{2} \rfloor), T(n, \lceil \frac{d}{2} \rceil))$$

↗
recursion

$$+ O(\log n)$$

↑ non-recursive matrix
mult

$$\Rightarrow T(n, d) \leq O(\log n \cdot \log d) \quad \square.$$

Lemma Can invert $n \times n$ triangular matrix in $O(\log^2 n)$ parallel time.

Proof Say we want to invert $\begin{pmatrix} A & 0 \\ B & C \end{pmatrix}$.

Inverse is $\begin{pmatrix} A^{-1} & 0 \\ -C^{-1}BA^{-1} & C^{-1} \end{pmatrix}$.

$$\left(\text{Check: } \begin{pmatrix} A & 0 \\ B & C \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ -C^{-1}BA^{-1} & C^{-1} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \right)$$

A and C are triangular.

$T(n)$ = parallel time to invert $n \times n$ ^{triangular} matrix

$$T(n) \leq T(n/2) + O(\log n)$$

$$\Rightarrow T(n) \leq O(\log^2 n). \quad \square$$

Theorem Can compute coeffs of char poly $p_A(t)$ in $O(\log^2 n)$ time.

Proof. Use Newton - Girard.

If $p_A(t) = t^n + c_{n-1}t^{n-1} + \dots + c_1t + c_0$

$$s_i = \text{trace}(A^i),$$

$$\text{then } \begin{pmatrix} c_{n-1} \\ c_{n-2} \\ \vdots \\ c_1 \\ c_0 \end{pmatrix} = \begin{pmatrix} -1 & & & & \\ s_1 & -2 & & & \\ s_2 & s_1 & -3 & & \\ s_3 & s_2 & s_1 & -4 & \\ & \vdots & & \ddots & \\ s_{n-1} & s_{n-2} & \dots & & \end{pmatrix}^{-1} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}$$

- Compute $s_1, \dots, s_n$ in $O(\log^2 n)$ parallel time
- Invert matrix in $O(\log^2 n)$ par. time
- Matrix-vector mult. in $O(\log n)$ par. time $\square$

Cor. Compute $\det(A)$ in $O(\log^2 n)$ par. time

Proof $c_0 = \det(A)$.

Corollary Compute A^{-1} in $O(\log^2 n)$ parallel time.

Proof Use Cayley-Hamilton theorem.

$$0 = p_A(A) = A^n + c_{n-1}A^{n-1} + \dots + c_1A + c_0I.$$

If A^{-1} exists, then

$$A^{-1} = \frac{-1}{c_0} (A^{n-1} + c_{n-1}A^{n-2} + \dots + c_1I)$$

($c_0 = \det(A)$ nonzero iff A invertible)

Corollary Given $f, g \in \mathbb{C}[x]$ of degree $\leq n$, can compute $\text{EES}(f, g)$ in $O(\log^2 n)$ parallel time.

Proof Last lecture, we saw $\text{EES}(f, g)$ reduces to linear algebra on matrices of size $\leq 2n \times 2n$. $\square$

Next time: More linear alg, faster $\text{res}(f, g)$