Lecture 21: Parallel Algorithms, Non-Uniform Algorithms & Linear Algebra in Parallel

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

November 25, 2020

Overview

- Administrivia
- Parallel Algorithms: Computational Model
- Linear Algebra in Fast Parallel Time
- Conclusion
- Acknowledgements

Rate this course!

Please log in to

https://evaluate.uwaterloo.ca/

from *November 24th until December 7th* and provide us with your evaluation and feedback on the course!

- This would really help me figuring out what worked and what didn't for the course
- And whether I should put memes or gifs into my slides...
- Teaching this course is also a learning experience for me :)

Research Opportunities at UW!

Consider doing a URA, URF or USRA with a U Waterloo faculty!
See research openings at:

- Undergraduate Research Assistanship (URA):

  `https://cs.uwaterloo.ca/computer-science/current-undergraduate-students/research-opportunities/undergraduate-research-assistantship-ura-program`

- Undergraduate Research Fellowship (URF):

  `https://grec.cs.uwaterloo.ca/`

- Undergraduate Research Internship (URI):

  `https://cs.uwaterloo.ca/current-undergraduate-students/research-opportunities/undergraduate-research-internship-uri-program`

- For Canadians, please check out NSERC's USRA:

  `https://cs.uwaterloo.ca/usra`

Parallel Model of Computation

- Our computer has an "unlimited" number of (identical) processors

Parallel Model of Computation

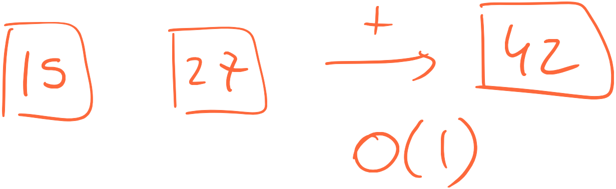
- Our computer has an "unlimited" number of (identical) processors
- Each processor has "unlimited" memory with unrestricted access

Parallel Model of Computation

- Our computer has an "unlimited" number of (identical) processors
- Each processor has "unlimited" memory with unrestricted access
- Or equivalently, they share "unlimited" memory (so, once one processor computes something, can assume everyone can use the result of computation)

Parallel Model of Computation

- Our computer has an "unlimited" number of (identical) processors
- Each processor has "unlimited" memory with unrestricted access
- Or equivalently, they share "unlimited" memory (so, once one processor computes something, can assume everyone can use the result of computation)
- Each processor can do *one* of the following operations in *unit time* (called a <u>step</u>):
  - *fetch* from memory
  - any binary operation (i.e. $+$, $-$, $\times$, $\div$, $\wedge$, $\vee$)
  - *storing* to memory



Parallel Model of Computation

- Our computer has an "unlimited" number of (identical) processors
- Each processor has "unlimited" memory with unrestricted access
- Or equivalently, they share "unlimited" memory (so, once one processor computes something, can assume everyone can use the result of computation)
- Each processor can do *one* of the following operations in *unit time* (called a step):
  - *fetch* from memory
  - any binary operation (i.e. $+$, $-$, $\times$, $\div$, $\wedge$, $\vee$)
  - *storing* to memory
- Before starting the computation, *entire input* stored in memory (thus accessible by all processors)

Parallel Model of Computation

- Our computer has an "unlimited" number of (identical) processors
- Each processor has "unlimited" memory with unrestricted access
- Or equivalently, they share "unlimited" memory (so, once one processor computes something, can assume everyone can use the result of computation)
- Each processor can do *one* of the following operations in *unit time* (called a step):
  - *fetch* from memory
  - any binary operation (i.e. $+, -, \times, \div, \wedge, \vee$)
  - *storing* to memory
- Before starting the computation, *entire input* stored in memory (thus accessible by all processors)
- Complexity Measures:
  - *Depth*: number of *sequential* steps needed from start to end.
  - *Width*: *maximum* number of processors needed in one "level" of the computation.
  - *Total number of operations*: total number of operations performed by all processors.

Non-Uniform Model of Computation

- Turing Machines are the model we are used to:

  A *fixed-length code* that can handle inputs *of any size*.

One algorithm to rule them (inputs) all!

Non-Uniform Model of Computation

- Turing Machines are the model we are used to:

  A *fixed-length code* that can handle inputs *of any size*.
- This is what you are used to write in your programming classes, co-ops.

Non-Uniform Model of Computation

- Turing Machines are the model we are used to:

  A *fixed-length code* that can handle inputs *of any size*.

- This is what you are used to write in your programming classes, co-ops.
- Non-Uniform Model of Computation:

  Specific code C_n which handles only inputs from Σ^n.

length n

$\Sigma \leftarrow$ alphabet

$(\{0,1\})$

Non-Uniform Model of Computation

- Turing Machines are the model we are used to:

  A *fixed-length code* that can handle inputs *of any size*.

- This is what you are used to write in your programming classes, co-ops.
- Non-Uniform Model of Computation:

  Specific code C_n which handles only inputs from Σ^n.

- In non-uniform model, your "code" is a family of codes $\{C_n\}_{n\geq 0}$ where C_n only handles input of size n

Non-Uniform Model of Computation

- Turing Machines are the model we are used to:

  A *fixed-length code* that can handle inputs *of any size*.

- This is what you are used to write in your programming classes, co-ops.
- Non-Uniform Model of Computation:

  Specific code C_n which handles only inputs from Σ^n.

- In non-uniform model, your "code" is a family of codes $\{C_n\}_{n\geq 0}$ where C_n only handles input of size n
- Is that a reasonable model of computation?
  - Used in design of VSLI circuits

    `https://en.wikipedia.org/wiki/Very_Large_Scale_Integration`

  - Used for any "special purpose" computations
- Such non-uniform codes are called *circuits*.

Parallel Model in Algebraic Setting

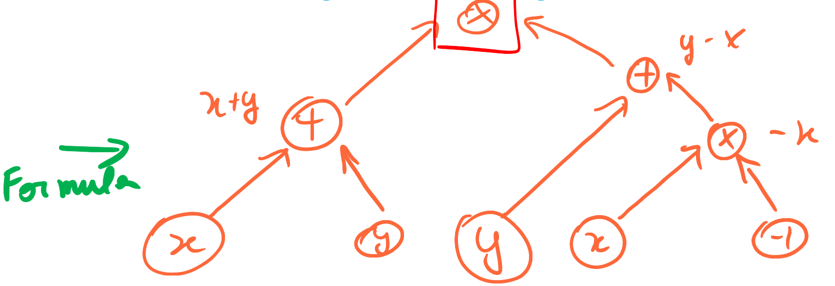
- For algebraic computations (multiplying two matrices, computing determinants, solving linear systems, solving polynomial equations...) it makes sense to try algorithms which *only use algebraic operations* (i.e. $+$, $-$, $\times$, $\div$)

arithmetic complexity

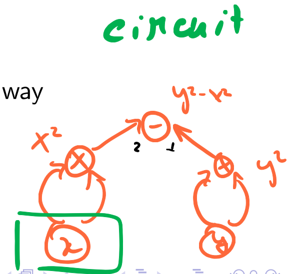
Parallel Model in Algebraic Setting

- For algebraic computations (multiplying two matrices, computing determinants, solving linear systems, solving polynomial equations...) it makes sense to try algorithms which *only use algebraic operations* (i.e. $+$, $-$, $\times$, $\div$)
- Algorithms in algebraic setting will compute:
  - polynomials
  - matrices
  - group elements
  - other algebraic objects

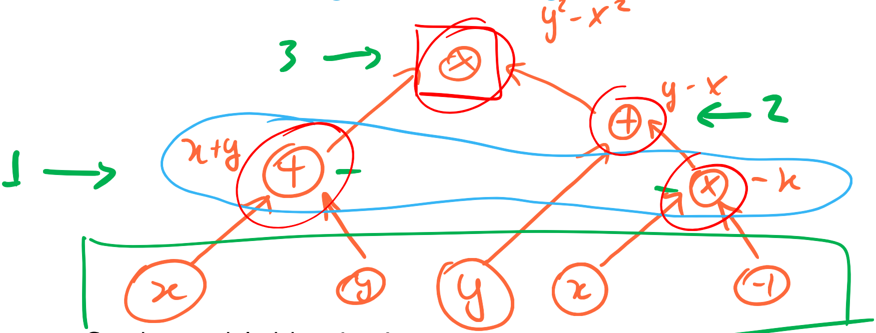
Parallel Model in Algebraic Setting



- Can be modeled by circuits:
  - Directed Acyclic Graphs
  - Each gate is either: input, algebraic operation
  - Non-input gates compute polynomial in natural way
  - Choose one gate to be the "output gate"



Parallel Model in Algebraic Setting



- Can be modeled by circuits:
  - Directed Acyclic Graphs
  - Each gate is either: input, algebraic operation
  - Non-input gates compute polynomial in natural way
  - Choose one gate to be the "output gate"
- Complexity measures:
  - *Depth*: length of longest path from input to output
  - *Width*: length of largest "layer"
  - *Operations*: total number of operations

Main Objects for Today's Class

1. Determinant:

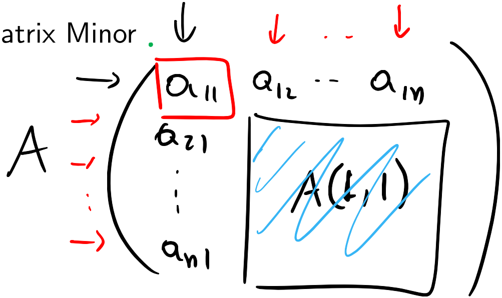
$$\det(X) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \cdot \prod_{i=1}^{n} x_{i\sigma(i)}$$

Main Objects for Today's Class

1. Determinant:

$$\det(X) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^{n} x_{i\sigma(i)}$$

2. Matrix Minor



square submatrices

Main Objects for Today's Class

1. Determinant:

$$\det(X) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^{n} x_{i\sigma(i)}$$

2. Matrix Minor
3. Matrix Adjugate: the matrix $Adj(X)$ is the unique matrix such that

$$Adj(X) \cdot X = \det(X) \cdot I_n$$

Main Objects for Today's Class

1. Determinant:

$$\det(X) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \cdot \prod_{i=1}^{n} x_{i\sigma(i)}$$

2. Matrix Minor
3. Matrix Adjugate: the matrix $Adj(X)$ is the unique matrix such that

$$Adj(X) \cdot X = \det(X) \cdot I_n$$

4. Inverse of matrix:

$$X^{-1} = \frac{1}{\det(X)} \cdot Adj(X)$$

Main Objects for Today's Class

1. Determinant:

$$\det(X) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \cdot \prod_{i=1}^{n} x_{i\sigma(i)}$$

2. Matrix Minor
3. Matrix Adjugate: the matrix $Adj(X)$ is the unique matrix such that

$$Adj(X) \cdot X = \det(X) \cdot I_n$$

4. Inverse of matrix:

$$X^{-1} = \frac{1}{\det(X)} \cdot Adj(X)$$

5. Characteristic polynomial of matrix:

encode useful info.

$$p_X(\lambda) = \det(X - \lambda \cdot I_n) = \sum_{i=0}^{n} p_{n-i}\lambda^i$$

$p_0 = p_X(0) = \det(X)$

Foundational Problems in Linear Algebra

- Computing the determinant of a matrix
- Solving a linear system of equations
- Inverting a matrix
- Computing the adjugate of a matrix
- Computing characteristic polynomial of matrix

Foundational Problems in Linear Algebra

- Computing the determinant of a matrix
- Solving a linear system of equations
- Inverting a matrix
- Computing the adjugate of a matrix
- Computing characteristic polynomial of matrix

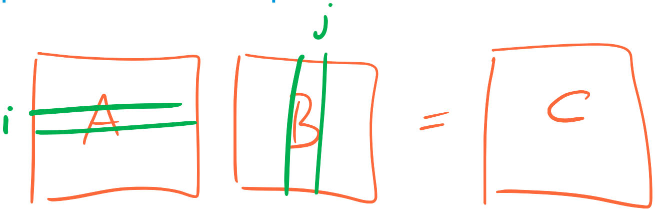
$N = n^2$

Can we solve the problems above fast in parallel time?

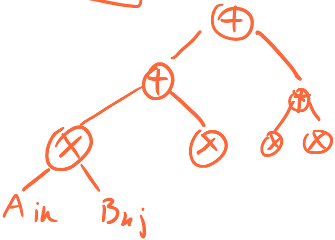
Fast here means: on input of size N, we have an algebraic circuit of

- size poly(N) (i.e. total number of operations)
- depth $\log^c(N)$, for some constant $c > 0$.

Example: Matrix Multiplication



$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$$

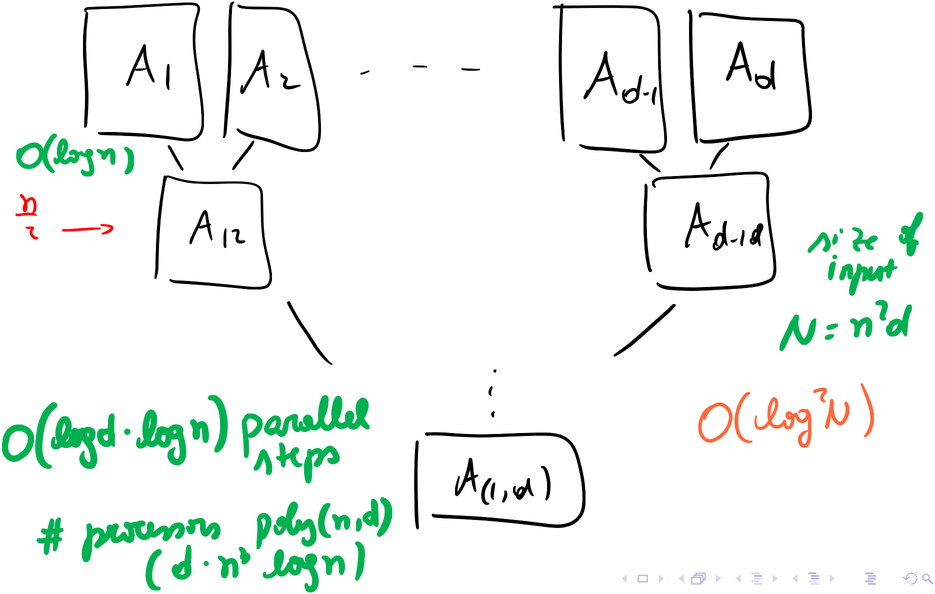


$$(A_{i1}, \ldots, A_{in}) \begin{pmatrix} B_{1j} \\ \vdots \\ B_{nj} \end{pmatrix}$$

$O(\log n)$ Parallel time

n^3 processors

Example: Multiplication of Sequence of Matrices



Our Task

1. Computing the determinant of a matrix
   (5) ⇒ (1)
2. Solving a linear system of equations
   (3) ⇒ (2)
3. Inverting a matrix
   (1) + (4) ⇒ (3)
4. Computing the adjugate of a matrix
   (5) ⇒ (4)
5. Computing characteristic polynomial of matrix ← TODAY

Can we solve the problems above fast in parallel time?

Fast here means: on input of size N, we have an algebraic circuit of

- size poly(N) (i.e. total number of operations)
- depth $\log^c(N)$, for some constant $c > 0$.

Cayley-Hamilton Theorem

Theorem (Cayley-Hamilton Theorem)

Let A be an $n \times n$ matrix, and $p_A(\lambda)$ be its characteristic polynomial. Then

$$p_A(A) = 0$$

Cayley-Hamilton Theorem

Theorem (Cayley-Hamilton Theorem)

Let A be an $n \times n$ matrix, and $p_A(\lambda)$ be its characteristic polynomial. Then

$$p_A(A) = 0$$

- From Cayley-Hamilton, we can extract the adjugate from characteristic polynomial!

Cayley-Hamilton Theorem

Theorem (Cayley-Hamilton Theorem)

Let A be an $n \times n$ matrix, and $p_A(\lambda)$ be its characteristic polynomial. Then

$$p_A(A) = 0$$

- From Cayley-Hamilton, we can extract the adjugate from characteristic polynomial!
- $$0 = p_A(A) = \det(A) \cdot I + \sum_{i=1}^{n} p_{n-i} \cdot A^i \Rightarrow$$
  $$\Rightarrow \det(A) \cdot I = -\sum_{i=1}^{n} p_{n-i} \cdot A^i$$
  $$= A \cdot \left(-\sum_{i=1}^{n} p_{n-i} \cdot A^{i-1} \right) = A \cdot Adj(A)$$

Cayley-Hamilton

all elements are multiples of A

by uniqueness

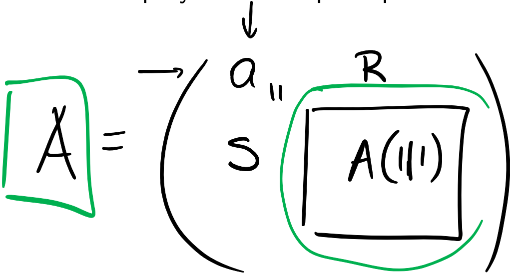
Adj(A)

Berkowitz's Algorithm - Outline

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.

Berkowitz's Algorithm - Outline

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$



Berkowitz's Algorithm - Outline

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$
3. Recurse!

Berkowitz's Algorithm - Outline

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$
3. Recurse!
4. Recursion can be encoded by sequence of matrix multiplications

Berkowitz's Algorithm - Outline

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$
3. Recurse!
4. Recursion can be encoded by sequence of matrix multiplications
5. Show that all matrices can be computed simultaneously

Berkowitz's Algorithm - Outline

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$
3. Recurse!
4. Recursion can be encoded by sequence of matrix multiplications
5. Show that all matrices can be computed simultaneously
6. By first part, know that can compute product of matrices in $O(\log^2 n)$ parallel time.

Useful Lemmas

Lemma

Let $A = \begin{pmatrix} a_{11} & R \\ S & M_1 \end{pmatrix}$. Then $= A(1|1)$

$$p_A(\lambda) = (a_{11} - \lambda) \cdot \det(M_1 - \lambda \cdot I) + R \cdot adj(M_1 - \lambda \cdot I) \cdot S$$

$p_{M_1}(\lambda)$

Can be computed from $p_{M_1}(\lambda)$

Can compute $p_A(\lambda)$ from $p_{M_1}(\lambda)$!

Useful Lemmas

Lemma

Let $A = \begin{pmatrix} a_{11} & R \\ S & M_1 \end{pmatrix}$. *Then*

$$p_A(\lambda) = (a_{11} - \lambda) \cdot \det(M_1 - \lambda \cdot I) + R \cdot adj(M_1 - \lambda \cdot I) \cdot S$$

$$\left(\begin{array}{c|c} X_{11} & X_{12} \\ \hline X_{21} & X_{22} \end{array}\right) =: X$$

$X_{21}X_{11}$ $X_{11}X_{12}$ $X_{22}X_{21}$ $X_{12}X_{22}$

X, X_{11}, X_{22} square

Lemma: $A = \left(\begin{array}{c|c} X_{11} & 0 \\ \hline * & X_{22} \end{array}\right)$ or $B = \left(\begin{array}{c|c} X_{11} & * \\ \hline 0 & X_{22} \end{array}\right)$

$\det(A) = \det(B) = \det(X_{11}) \det(X_{22})$

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} I & 0 \\ -X_{22}^{-1}X_{21} & X_{22}^{-1} \end{pmatrix} = \begin{pmatrix} X_{11} - X_{12}X_{22}^{-1}X_{21} & X_{12}X_{22}^{-1} \\ 0 & I \end{pmatrix}$$

Useful Lemmas

Lemma

Let $A = \begin{pmatrix} a_{11} & R \\ S & M_1 \end{pmatrix}$. *Then*

$$A - \lambda I = \begin{pmatrix} a_{11} - \lambda & R \\ S & M_1 - \lambda I \end{pmatrix}$$

$$p_A(\lambda) = (a_{11} - \lambda) \cdot \det(M_1 - \lambda \cdot I) + R \cdot adj(M_1 - \lambda \cdot I) \cdot S$$

$$\det(X) = \det(X_{22}) \cdot \det(X_{11} - X_{12} X_{22}^{-1} X_{21})$$

$X_{11} = a_{11} - \lambda \quad X_{12} = R \quad X_{21} = S \quad X_{22} = M_1 - \lambda I$

$$\underbrace{\det(A - \lambda I)}_{p_A(\lambda)} = \det(M_1 - \lambda I) \det(\underbrace{(a_{11} - \lambda) - R(M_1 - \lambda I)^{-1} S}_{1 \times 1})$$

$$= (a_{11} - \lambda) \cdot \det(M_1 - \lambda I) - R \, adj(M_1 - \lambda I) S$$

∎

Useful Lemmas

Computing Adj by using char. poly

Lemma

Let M be an $(n-1)\times(n-1)$ matrix and $p_M(\lambda)=\sum_{i=0}^{n-1} q_{n-1-i}\lambda^i$. Then

$$adj(M-\lambda\cdot I) = -\sum_{k=2}^{n}(M^{k-2}q_0+\cdots+I\cdot q_{k-2})\cdot\lambda^{n-k}$$

CH $\Rightarrow p_M(M)=0$

$$(M-\lambda I)\cdot adj(M-\lambda I) = \underbrace{\det(M-\lambda I)}_{p_M(\lambda)}\cdot I_{n-1}$$

$$(M-\lambda I)\cdot RHS = -\sum_{k=2}^{n}(M^{k-1}q_0+M^{k-2}q_1+\cdots+Mq_{k-2})\lambda^{n-k}$$

$$+\sum_{k=1}^{n-1}(M^{k-1}q_0+\cdots+Iq_{k-1})\lambda^{n-k} = \sum_{k=1}^{n-1} I\cdot q_{k-1}\cdot\lambda^{n-k}$$

$$(M^{k-1}q_0+M^{k-2}q_1+\cdots+Mq_{k-2}+Iq_{k-1})$$

$$(M - \lambda I)\,\text{RHS} = I \cdot \underbrace{\left(\sum_{k=1}^{n-1} q_{k-1} \lambda^{n-k}\right)}_{p_M(\lambda)}$$

$$j = n-1-k$$

$$\cancel{(M - \lambda I)}\,\text{Adj}(M - \lambda I) = \cancel{(M - \lambda I)} \cdot \text{RHS}$$

Computing Characteristic Polynomial as Matrix Multiplications

(i.e. put two lemmas together)

$$p_A(\lambda) = (a_{11}-\lambda)\cdot p_M(\lambda) - R\left(\sum_{k=2}^{n}\left(M^{k-2}q_0 + \cdots + I\,q_{k-2}\right)\lambda^{n-k}\right)S$$

coeff. of $p_M(\lambda)$

$$\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_i \\ \vdots \\ p_n \end{pmatrix} = \underbrace{\begin{pmatrix} & & \\ & I & \\ & & \end{pmatrix}}_{(n+1)\times n} \begin{pmatrix} q_0 \\ \vdots \\ q_{n-1} \end{pmatrix}$$

i

p_i is the coefficient of λ^{n-i}

$$p_A(\lambda) = \sum p_{n-i}\lambda^i$$

Computing Characteristic Polynomial as Matrix Multiplications

$$p_A(\lambda) = (a_{11} - \lambda) \cdot p_u(\lambda) -$$

$$R\left(\sum_{k=2}^{n}\left(M^{k-2} q_0 + \cdots + I q_{k-2}\right)\lambda^{n-k}\right)S$$

$b_i = RM^iS$ (number 1×1)

k^{th} row of T_1 is the coeff. of λ^{n-k} in expression above:

$$a_{11}q_{k-1} - q_k \quad -\left(b_{k-2}\cdot q_0 + b_{k-3}q_1 + \cdots + b_0 q_{k-2}\right)$$

Computing Characteristic Polynomial as Matrix Multiplications

$$a_{11} q_{k-1} - q_k \cdot 1 - \left(b_{k-2} \cdot q_0 + b_{k-3} q_1 + \cdots + b_0 q_{k-2}\right)$$

$$\Big(\underbrace{-b_{k-2}\ \ -b_{k-3}\ \cdots\ -b_0\ \ a_{11}\ \ -1}_{k+1}\ \ \underbrace{0\ \cdots\ 0}_{(n+1)-(k+1) = n-k}\Big) \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_{n-1} \end{pmatrix} = p_k$$

Putting things together

$$\underbrace{\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{pmatrix}}_{p_A(\lambda)} = \underbrace{\begin{pmatrix} -1 & 0 & \cdots & & & 0 \\ a_{11} & -1 & 0 & \cdots & & 0 \\ -b_0 & a_{11} & -1 & 0 & \cdots & 0 \\ -b_1 & -b_0 & a_{11} & -1 & 0 \cdots & 0 \\ \vdots & & -b_0 & a_{11} & \ddots & \\ & & & \ddots & \ddots & -1 \\ -b_{n-2} & \cdots & & & -b_0 & a_{11} \end{pmatrix}}_{T_1} \underbrace{\begin{pmatrix} q_0 \\ \vdots \\ q_{n-1} \end{pmatrix}}_{p_{M_1}(\lambda)}$$

in $O(\log^2 n)$ parallel time

to compute T_1 we need to compute b_i = RM's ← can compute fast in parallel

Putting things together

$O(\log^2 n)$

$$\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{pmatrix} = \underbrace{T_1 \cdot T_2 \cdots T_n}_{\text{product of matrices}}$$

$T_i = (n+2-i) \times (n+1-i)$

matrix

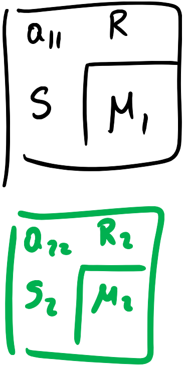
Computing T_i matrices simultaneously

to compute T_1

we needed. for compute

$$b_i = R \cdot M_1^{-1} S$$

input (R), also input (M_1), input (S)



Analysis of Algorithm

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.

$$\begin{pmatrix} p_0 \\ \vdots \\ p_n \end{pmatrix} = T_1 T_2 \cdots T_n$$

Analysis of Algorithm

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$

Analysis of Algorithm

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$
3. Recurse!

Analysis of Algorithm

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$
3. Recurse!
4. Recursion can be encoded by sequence of matrix multiplications

Analysis of Algorithm

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$
3. Recurse!
4. Recursion can be encoded by sequence of matrix multiplications
5. Show that all matrices can be computed simultaneously

Analysis of Algorithm

1. Strategy: compute characteristic polynomial via sequence of matrix multiplications.
2. Reduce computation of characteristic polynomial of A to computation of characteristic polynomial of principal minor $A(1 \mid 1)$
3. Recurse!
4. Recursion can be encoded by sequence of matrix multiplications
5. Show that all matrices can be computed simultaneously
6. By first part, know that can compute product of matrices in $O(\log^2 n)$ parallel time.

$O(\log^2 n) + O(\log^2 n) = O(\log^2 n)$

all T_i's — product of T_i's

Conclusion

- Parallel algorithms are important for many applications, when we have access to a lot of computing power, but want answers fast.

Conclusion

- Parallel algorithms are important for many applications, when we have access to a lot of computing power, but want answers fast.
- Applications in
  - Scientific Computing
  - Linear Algebra Computations
    - Machine Learning
  - many more...

Conclusion

- Parallel algorithms are important for many applications, when we have access to a lot of computing power, but want answers fast.
- Applications in
  - Scientific Computing
  - Linear Algebra Computations
    - Machine Learning
  - many more...
- *Parallel model of computation*: different model from usual (sequential) algorithms, non-uniform.

Conclusion

- Parallel algorithms are important for many applications, when we have access to a lot of computing power, but want answers fast.
- Applications in
  - Scientific Computing
  - Linear Algebra Computations
    - Machine Learning
  - many more...
- *Parallel model of computation*: different model from usual (sequential) algorithms, non-uniform.
- Saw how to compute foundational objects from linear algebra in parallel time.

Acknowledgement

- Lecture based largely on:
  - Csanky's paper [Csanky 1976]
  - Berkowitz's paper [Berkowitz 1984]
- For combinatorial interpretation of Berkowitz's algorithm, see

  `https://arxiv.org/pdf/math/0201315.pdf`

References I

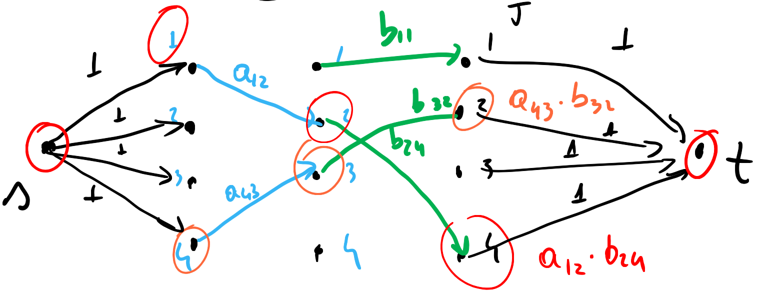
Berkowitz, S. J. (1984).
On computing the determinant in small parallel time using a small number of processors.
Information processing letters, 18(3), 147-150.

Csanky, L. (1975).
Fast parallel matrix inversion algorithms.
In 16th Annual Symposium on Foundations of Computer Science (FOCS) (pp. 11-12). IEEE.

ABP (Algebraic Branching Program)

$$\sum a_{ik} \cdot b_{ki} = tr\left(A \cdot B \underbrace{\begin{pmatrix}1\\ \vdots \\ 1\end{pmatrix}(1 \cdots 1)}_{J}\right)_{n \times n}$$

$n = 4$



$$\sum (\text{paths } s \to t) \qquad 1 \cdot a_{12} \cdot b_{24} \cdot 1$$

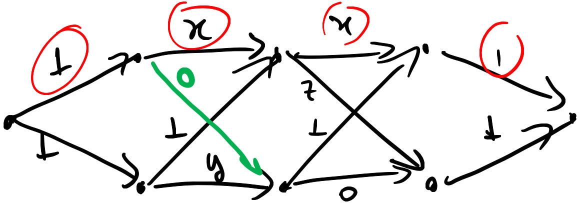
$$\sum a_{ik} b_{kj}$$

today

$$\operatorname{tr}(X_1 X_2 \cdots X_d) = p(x)$$

$\updownarrow$

ABP computing $p(x)$



$$\operatorname{tr}\left(\begin{pmatrix} x & 0 \\ 1 & y \end{pmatrix}\begin{pmatrix} x & z \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\right)$$