Parallelizing Time With Polynomial Circuits

Ryan Williams

Carnegie Mellon University

and

Microsoft Research (Intern)

RED GREEN BLUE ORANGE

Parallelism and Serialism: Some Questions

Parallelism and Serialism: Some Questions

- To what extent can arbitrary serial computations be parallelized?

Parallelism and Serialism: Some Questions

- To what extent can arbitrary serial computations be parallelized?
- To what extent can serial algorithms running in time $t(n)$ be parallelized?

  – *Assuming a robust serial machine model.*

Parallelism and Serialism: Some Questions

- To what extent can arbitrary serial computations be parallelized?
- To what extent can serial algorithms running in time $t(n)$ be parallelized?

  – *Assuming a robust serial machine model.*
- What is the smallest $f(n) = o(n)$ such that a serial time t algorithm can be represented by a circuit of depth at most $f(t)$?

  – *Note could always have an exponential sized circuit of $O(1)$ depth and unbounded fan-in*

Parallelism and Serialism: Some Questions

- To what extent can arbitrary serial computations be parallelized?
- To what extent can serial algorithms running in time $t(n)$ be parallelized?

  – *Assuming a robust serial machine model.*
- What is the smallest $f(n) = o(n)$ such that a serial time t algorithm can be represented by a circuit of depth at most $f(t)$?

  – *Note could always have an exponential sized circuit of $O(1)$ depth and unbounded fan-in*
- What is the smallest f such that a serial time t algorithm can be represented by a $t(n)^{O(1)}$*-size circuit of depth at most* $f(t(n))$?

  – **Can circuits with only $poly(t)$ gates speed up serial time t computations?**

Can circuits with only $poly(t)$ gates speed up serial time t computations?

This is the question we address.

Can circuits with only $poly(t)$ gates

speed up serial time t computations?

This is the question we address.

Short Answer:

YES,

... but some gates in the circuits have unbounded fan-in

Prior Work, in brief

Relatively old area

- All prior work for general computational models focused on extending Hopcroft-Paul-Valiant's seminal and deep result:

$$\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$$

Prior Work, in brief

Relatively old area

- All prior work for general computational models focused on extending Hopcroft-Paul-Valiant's seminal and deep result:

$$\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$$

Main Idea of $\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$**:**

Prior Work, in brief

Relatively old area

- All prior work for general computational models focused on extending Hopcroft-Paul-Valiant's seminal and deep result:

$$\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$$

Main Idea of $\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$**:**

Divide-and-conquer on Computation Graph:

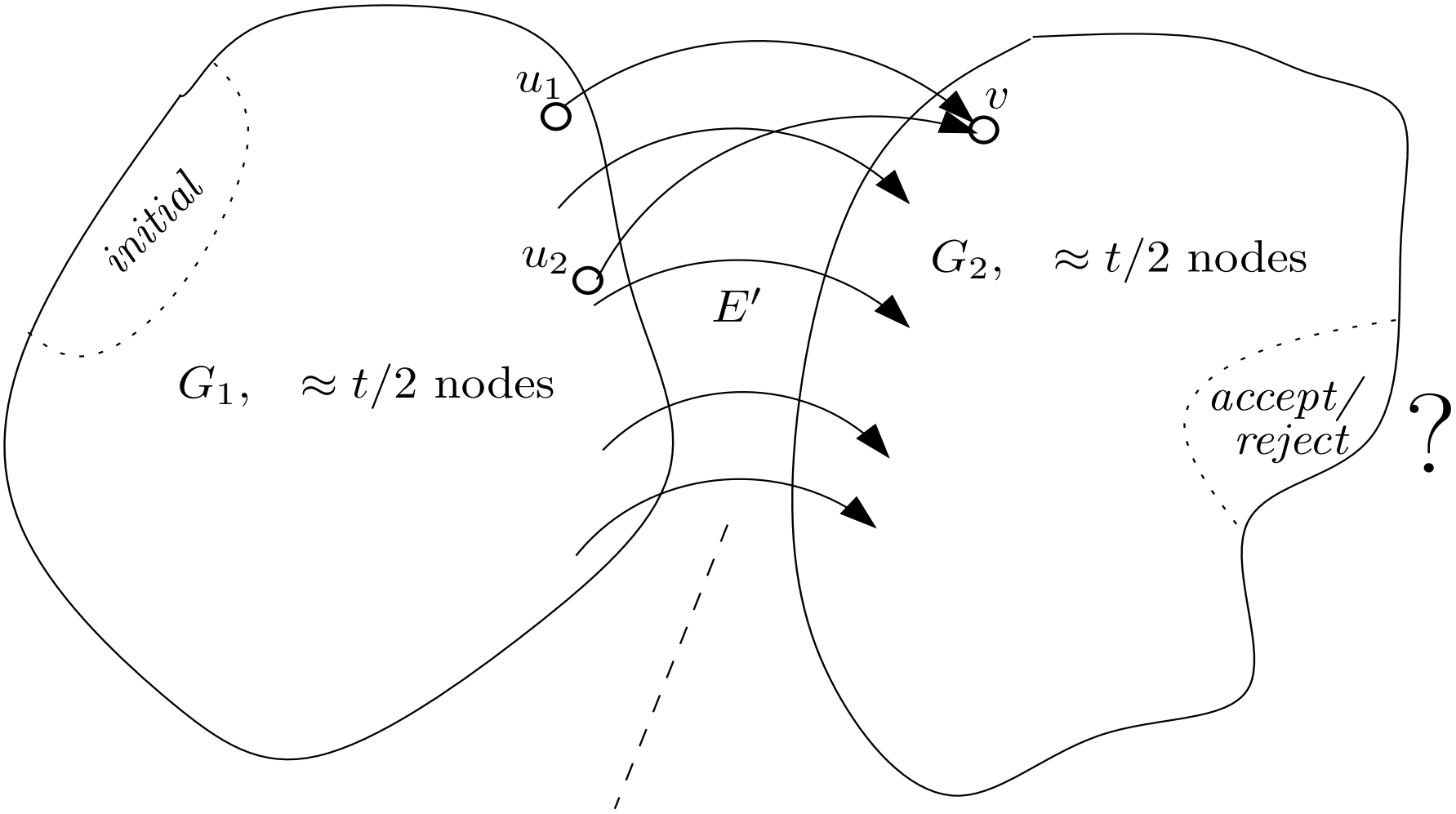
DAG with a node for each timestep (or block of timesteps)

– *Value* of node i is state and symbols read/written in step(s) corresponding to i

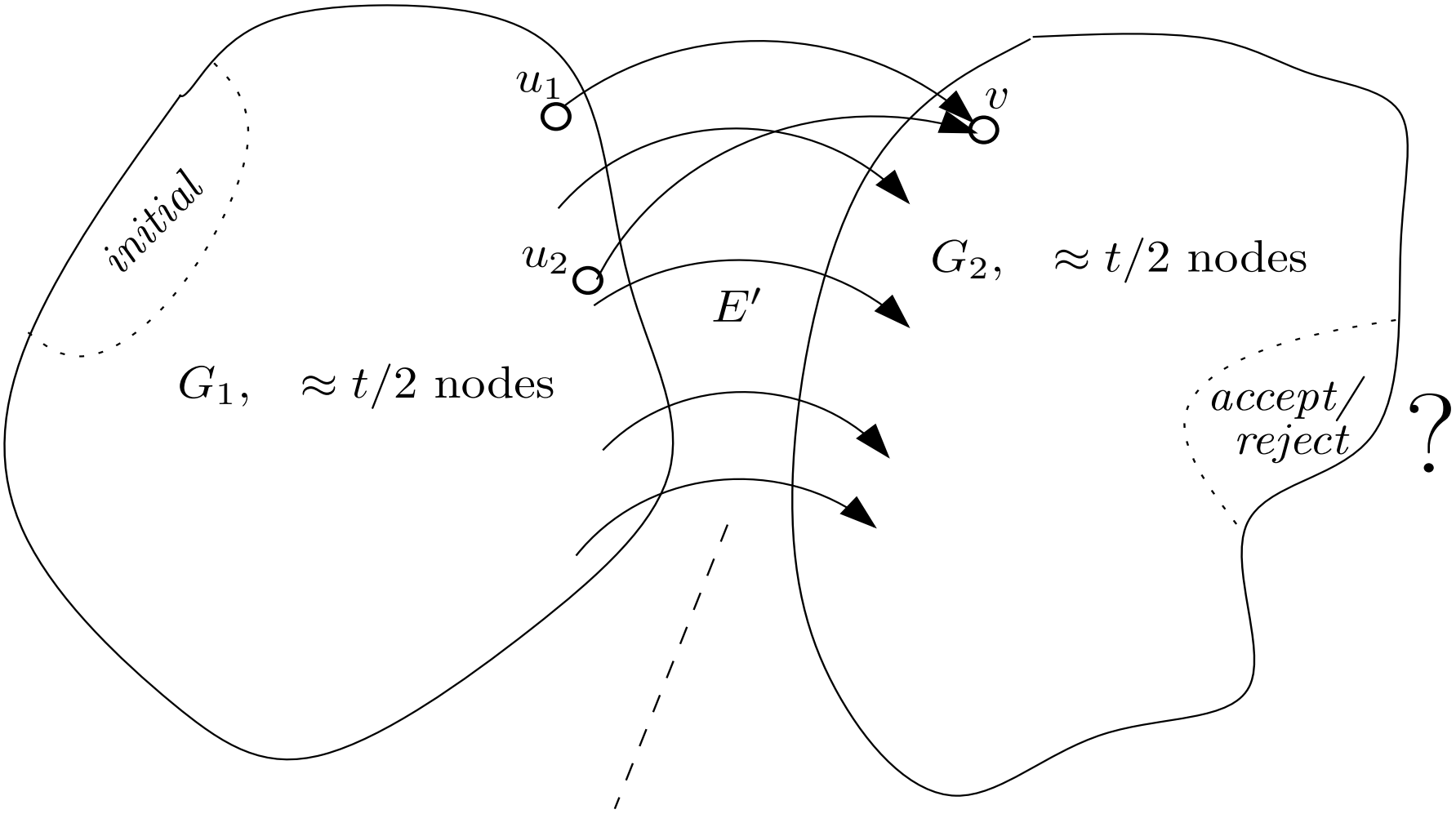
Arcs: represent read/write/state dependencies

Main Idea of $\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$

Main Idea of $\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$

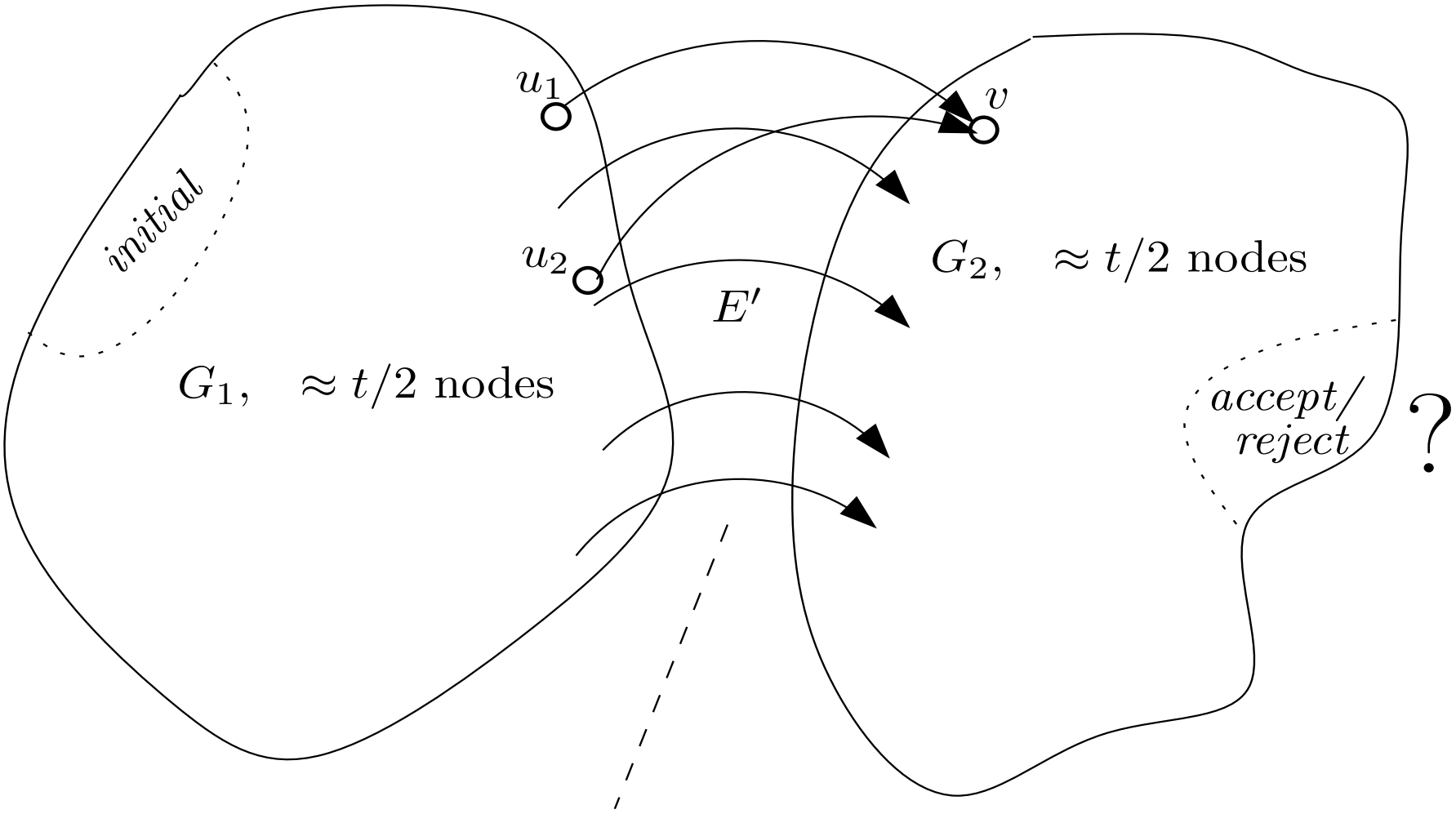


Main Idea of $\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$



$|E'| \leq t/\log t \Longrightarrow$ "Guess" values at endpoints of E', recurse on G_1 and G_2 separately

Main Idea of $\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[t(n)/\log t(n)]$



$|E'| \leq t/\log t \Longrightarrow$ "Guess" values at endpoints of E', recurse on G_1 and G_2 separately

$|E'| > t/\log t \Longrightarrow$ To get value of v in G_2, recurse on G_1 for value of u_1 and u_2

Known: Big barriers to this approach

Lower bounds on how well divide-and-conquer can do

E.g. To get $o(t)$ depth circuits, need a *super-polynomial* number of gates

Known: Big barriers to this approach

Lower bounds on how well divide-and-conquer can do

E.g. To get $o(t)$ depth circuits, need a *super-polynomial* number of gates

Intuition:

Too much information to be stored when one tries to guess all possible E' in parallel!

Random Access TM Model

(The serial model we're going to simulate in parallel)

Random Access TM Model

(The serial model we're going to simulate in parallel)

Have:

- Registers storing $O(1)$ bits each
- Index tape I of $O(\log t)$ bits

Random Access TM Model

(The serial model we're going to simulate in parallel)

Have:

- Registers storing $O(1)$ bits each
- Index tape I of $O(\log t)$ bits

In a single step, can:

1. Modify a bit of I
2. Read corresponding register
3. Write to corresponding register
4. Change state

Robust model – can simulate log-cost RAMs (unbounded registers) with constant factor overhead

Main Result

Time $t(n)$ random access TMs can be simulated by a

$t^{O(1)}$-size circuit family of depth $O(t/\log t)$.

(Note: More general result in paper; can tradeoff depth and size)

Main Result

Time $t(n)$ random access TMs can be simulated by a

$t^{O(1)}$-size circuit family of depth $O(t/\log t)$.

(Note: More general result in paper; can tradeoff depth and size)

Bad news: Some of the circuit's gates have unbounded fan-in.

Main Result

Time $t(n)$ random access TMs can be simulated by a

$t^{O(1)}$-size circuit family of depth $O(t/\log t)$.

(Note: More general result in paper; can tradeoff depth and size)

Bad news: Some of the circuit's gates have unbounded fan-in.

Good news: Construction is *uniform* –

Gives explicit, efficiently constructed circuits.

Our Approach:

A less extreme type of divide-and-conquer

Our Approach:

A less extreme type of divide-and-conquer

1. Partition computation into $O(t/\log t)$ blocks
2. Each block represents $O(\log t)$ consecutive steps

   Blocks are succinctly representable: $O(\log t)$ bits

Our Approach:

A less extreme type of divide-and-conquer

1. Partition computation into $O(t/\log t)$ blocks
2. Each block represents $O(\log t)$ consecutive steps

   Blocks are succinctly representable: $O(\log t)$ bits
3. Processing of single block is possible in $O(1)$ depth and $poly(t)$ size:
   - "String" these together

     $\Longrightarrow O(t/\log t)$ depth over all blocks

Local Actions and Blocks

Define **local action** of $A(x)$ at step i to be $\ell = (i, r, I)$, where

r is transition, I is description of index tape at step i

- Note $|\ell| = O(\log t)$

Local Actions and Blocks

Define **local action** of $A(x)$ at step i to be $\ell = (i, r, I)$, where

r is transition, I is description of index tape at step i

- Note $|\ell| = O(\log t)$

Define $S_{A(x)}$ to be the (unique) string of the form

$$\ell_0 \, \vec{r}_0 \, \ell_1 \, \vec{r}_1 \, \cdots \, \ell_{(t/\log t)-1} \, \vec{r}_{(t/\log t)-1},$$

Local Actions and Blocks

Define **local action** of $A(x)$ at step i to be $\ell = (i, r, I)$, where

r is transition, I is description of index tape at step i

- Note $|\ell| = O(\log t)$

Define $S_{A(x)}$ to be the (unique) string of the form

$$\ell_0 \, \vec{r}_0 \, \ell_1 \, \vec{r}_1 \, \cdots \, \ell_{(t/\log t)-1} \, \vec{r}_{(t/\log t)-1},$$

where ℓ_i is local action of $A(x)$ at step $i \cdot (\log t) + 1$,

Local Actions and Blocks

Define **local action** of $A(x)$ at step i to be $\ell = (i, r, I)$, where

r is transition, I is description of index tape at step i

- Note $|\ell| = O(\log t)$

Define $S_{A(x)}$ to be the (unique) string of the form

$$\ell_0 \, \vec{r}_0 \, \ell_1 \, \vec{r}_1 \, \cdots \, \ell_{(t/\log t)-1} \, \vec{r}_{(t/\log t)-1},$$

where ℓ_i is local action of $A(x)$ at step $i \cdot (\log t) + 1$,

$\vec{r}_i$ is vector of $(\log t) - 1$ transitions taken, starting at step $i \cdot (\log t) + 2$.

Local Actions and Blocks

Define **local action** of $A(x)$ at step i to be $\ell = (i, r, I)$, where

r is transition, I is description of index tape at step i

- Note $|\ell| = O(\log t)$

Define $S_{A(x)}$ to be the (unique) string of the form

$$\ell_0 \, \vec{r}_0 \, \ell_1 \, \vec{r}_1 \, \cdots \, \ell_{(t/\log t)-1} \, \vec{r}_{(t/\log t)-1},$$

where ℓ_i is local action of $A(x)$ at step $i \cdot (\log t) + 1$,

$\vec{r}_i$ is vector of $(\log t) - 1$ transitions taken, starting at step $i \cdot (\log t) + 2$.

I.e. Write $O(t/\log t)$ local actions that are $O(\log t)$ steps apart from each other. Then add in the transitions for the missing steps.

Local Actions and Blocks

Define **local action** of $A(x)$ at step i to be $\ell = (i, r, I)$, where

r is transition, I is description of index tape at step i

- Note $|\ell| = O(\log t)$

Define $S_{A(x)}$ to be the (unique) string of the form

$$\ell_0 \, \vec{r}_0 \, \ell_1 \, \vec{r}_1 \, \cdots \, \ell_{(t/\log t)-1} \, \vec{r}_{(t/\log t)-1},$$

where ℓ_i is local action of $A(x)$ at step $i \cdot (\log t) + 1$,

$\vec{r}_i$ is vector of $(\log t) - 1$ transitions taken, starting at step $i \cdot (\log t) + 2$.

I.e. Write $O(t/\log t)$ local actions that are $O(\log t)$ steps apart from each other. Then add in the transitions for the missing steps.

- $|S_{A(x)}| = O(t)$

Local Actions and Blocks

Define **local action** of $A(x)$ at step i to be $\ell = (i, r, I)$, where

r is transition, I is description of index tape at step i

- Note $|\ell| = O(\log t)$

Define $S_{A(x)}$ to be the (unique) string of the form

$$\ell_0 \, \vec{r}_0 \, \ell_1 \, \vec{r}_1 \, \cdots \, \ell_{(t/\log t)-1} \, \vec{r}_{(t/\log t)-1},$$

where ℓ_i is local action of $A(x)$ at step $i \cdot (\log t) + 1$,

$\vec{r}_i$ is vector of $(\log t) - 1$ transitions taken, starting at step $i \cdot (\log t) + 2$.

I.e. Write $O(t/\log t)$ local actions that are $O(\log t)$ steps apart from each other. Then add in the transitions for the missing steps.

- $|S_{A(x)}| = O(t)$

Define a *block* to be an $\ell_i \, \vec{r}_i$ substring of $S_{A(x)}$.

Sketch of the Simulating Circuit

Circuit on input x constructs $S_{A(x)}$ in parallel, one block at a time

Sketch of the Simulating Circuit

Circuit on input x constructs $S_{A(x)}$ in parallel, one block at a time

Circuit implements two procedures, VERIFY and LAST-WRITE:

Sketch of the Simulating Circuit

Circuit on input x constructs $S_{A(x)}$ in parallel, one block at a time

Circuit implements two procedures, VERIFY and LAST-WRITE:

1. VERIFY$(\ell\vec{r}, i)$ accepts $\iff$ $\ell\vec{r}$ is the ith block of $S_{A(x)}$, and

2. LAST-WRITE(I, i, σ) accepts $\iff$

symbol σ is written at register I in the most recent timestep (over blocks $1, \ldots, i$) where register I is accessed

Sketch of the Simulating Circuit

Circuit on input x constructs $S_{A(x)}$ in parallel, one block at a time

Circuit implements two procedures, VERIFY and LAST-WRITE:

1. VERIFY$(\ell\vec{r}, i)$ accepts $\iff$ $\ell\vec{r}$ is the ith block of $S_{A(x)}$, and

2. LAST-WRITE(I, i, σ) accepts $\iff$

symbol σ is written at register I in the most recent timestep (over blocks $1, \ldots, i$) where register I is accessed

Fairly involved procedures

They invoke a number of deterministic logspace **checks** on a block

Sketch of the Simulating Circuit

Circuit on input x constructs $S_{A(x)}$ in parallel, one block at a time

Circuit implements two procedures, VERIFY and LAST-WRITE:

1. VERIFY$(\ell\vec{r}, i)$ accepts $\iff$ $\ell\vec{r}$ is the ith block of $S_{A(x)}$, and

2. LAST-WRITE(I, i, σ) accepts $\iff$

symbol σ is written at register I in the most recent timestep (over blocks $1, \ldots, i$) where register I is accessed

Fairly involved procedures

They invoke a number of deterministic logspace **checks** on a block

$\Longrightarrow$ Simulate a **check** with an $O(\log^2 t)$ depth, $O(poly(n))$ size circuit

Sketch of the Simulating Circuit

Circuit on input x constructs $S_{A(x)}$ in parallel, one block at a time

Circuit implements two procedures, VERIFY and LAST-WRITE:

1. VERIFY$(\ell\vec{r}, i)$ accepts $\iff$ $\ell\vec{r}$ is the ith block of $S_{A(x)}$, and

2. LAST-WRITE(I, i, σ) accepts $\iff$

symbol σ is written at register I in the most recent timestep (over blocks $1, \ldots, i$) where register I is accessed

Fairly involved procedures

They invoke a number of deterministic logspace **checks** on a block

$\Longrightarrow$ Simulate a **check** with an $O(\log^2 t)$ depth, $O(poly(n))$ size circuit

Can guess a **check**'s answer, then

Run the **check** AND Continue simulation in parallel

Sketch of the Simulating Circuit

Circuit on input x constructs $S_{A(x)}$ in parallel, one block at a time

Circuit implements two procedures, VERIFY and LAST-WRITE:

1. VERIFY$(\ell\vec{r}, i)$ accepts $\iff$ $\ell\vec{r}$ is the ith block of $S_{A(x)}$, and

2. LAST-WRITE(I, i, σ) accepts $\iff$

symbol σ is written at register I in the most recent timestep (over blocks $1, \ldots, i$) where register I is accessed

Fairly involved procedures

They invoke a number of deterministic logspace **checks** on a block

$\Longrightarrow$ Simulate a **check** with an $O(\log^2 t)$ depth, $O(poly(n))$ size circuit

Can guess a **check**'s answer, then

Run the **check** AND Continue simulation in parallel

$\Longrightarrow$ Checks don't contribute to overall depth by more than a constant

Sketch of VERIFY$(\ell\vec{r}, i)$**:**

(Top-down description, starting from *output* gate)

Use an AND to simultaneously pick a transition in $\vec{r}$, AND pick ℓ

Sketch of VERIFY$(\ell\vec{r}, i)$**:**

(Top-down description, starting from *output* gate)

Use an AND to simultaneously pick a transition in $\vec{r}$, AND pick ℓ

If ℓ is picked:

- Use OR to guess the $(i-1)$th block $\ell'\vec{r'}$,
- Call VERIFY$(\ell'\vec{r'}, i-1)$ to **check** state and index tape of ℓ.

  Call LAST-WRITE to **check** that the symbol read in ℓ is correct.

Sketch of VERIFY$(\ell\vec{r}, i)$**:**

(Top-down description, starting from *output* gate)

Use an AND to simultaneously pick a transition in $\vec{r}$, AND pick ℓ

If ℓ is picked:

- Use OR to guess the $(i-1)$th block $\ell'\vec{r'}$,
- Call VERIFY$(\ell'\vec{r'}, i-1)$ to **check** state and index tape of ℓ.

  Call LAST-WRITE to **check** that the symbol read in ℓ is correct.

If the jth component r_j of $\vec{r}$ is picked:

- Use OR to guess I, index tape for the step
- Use ℓ and $\vec{r}$ to **check** that the state of r_j, I are correct
- Call LAST-WRITE on I to **check** that symbol claimed to be read in r_j is correct

Sketch of LAST-WRITE(I, i, σ)**:**

- Use OR to guess ith block: $\ell_i \vec{r_i}$.
- Use AND to simultaneously:
  1. Call VERIFY$(\ell_i \vec{r_i}, i)$ (ensure block is correct), and
  2. **Check** if index tape is ever I in the block;

     if so, then verify σ is written,

     if not, then call LAST-WRITE$(I, i-1, \sigma)$.

Sketch of LAST-WRITE(I, i, σ)**:**

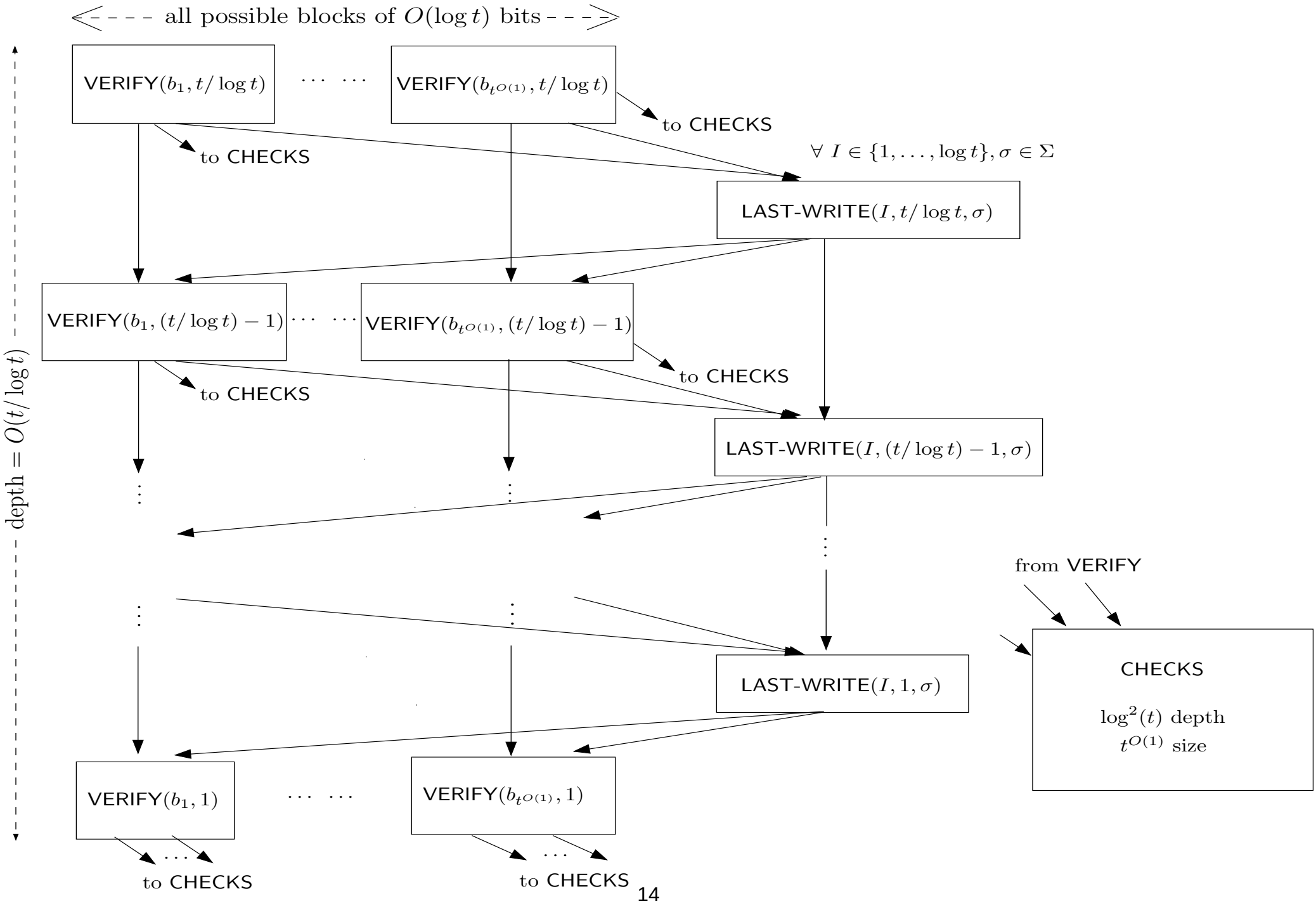
- Use OR to guess ith block: $\ell_i \vec{r_i}$.
- Use AND to simultaneously:
  1. Call VERIFY$(\ell_i \vec{r_i}, i)$ (ensure block is correct), and
  2. **Check** if index tape is ever I in the block;

     if so, then verify σ is written,

     if not, then call LAST-WRITE$(I, i-1, \sigma)$.

Observations:

- Constant number of OR/AND switches between two recursive calls.
- Depth of recursion = $O(t/\log t)$


all possible blocks of $O(\log t)$ bits
depth $= O(t/\log t)$
VERIFY$(b_1, t/\log t)$
VERIFY$(b_{t^{O(1)}}, t/\log t)$
to CHECKS
$\forall\ I \in \{1, \ldots, \log t\}, \sigma \in \Sigma$
LAST-WRITE$(I, t/\log t, \sigma)$
VERIFY$(b_1, (t/\log t) - 1)$
VERIFY$(b_{t^{O(1)}}, (t/\log t) - 1)$
LAST-WRITE$(I, (t/\log t) - 1, \sigma)$
LAST-WRITE$(I, 1, \sigma)$
VERIFY$(b_1, 1)$
VERIFY$(b_{t^{O(1)}}, 1)$
from VERIFY
CHECKS
$\log^2(t)$ depth
$t^{O(1)}$ size

Implications for Parallel Simulations

Corollary. Every log-cost time t RAM can be simulated by a log-cost CRCW PRAM in $O(t/\log t)$ time with $t^{O(1)}$ processors.

Previous parallel simulations required $2^{\Omega(t/\log t)}$ processors

Conclusion

Allow some unbounded fan-in

$\Longrightarrow$ Get $poly(t)$ size, $O(t/\log t)$ depth circuits, for time t algorithms.

Conclusion

Allow some unbounded fan-in

$\Longrightarrow$ Get $poly(t)$ size, $O(t/\log t)$ depth circuits, for time t algorithms.

- Can the same be done in the case of bounded fan-in?
- Can we improve upon $t/\log t$ depth?

Conclusion

Allow some unbounded fan-in

$\Longrightarrow$ Get $poly(t)$ size, $O(t/\log t)$ depth circuits, for time t algorithms.

- Can the same be done in the case of bounded fan-in?
- Can we improve upon $t/\log t$ depth?

Perhaps possible –

combine our ideas with Hopcroft-Paul-Valiant divide-and-conquer(?)